UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ANDREY LUIS TIETBOHL PALMA

A Clustering-Based Approach for Discovering Interesting Places in Trajectories

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Prof. Dr. Luis Otávio Álvares
Advisor

Prof. Dra. Vânia Bogorny
Coadvisor

Porto Alegre, December 2008

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Palma, Andrey Luis Tietbohl

A Clustering-Based Approach for Discovering Interesting Places in Trajectories / Andrey Luis Tietbohl Palma. – Porto Alegre: PPGC da UFRGS, 2008.

78 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2008. Advisor: Luis Otávio Álvares; Coadvisor: Vânia Bogorny.

1.Spatio-temporal 2.Clustering 3.CB-SMoT 4.Stops 5.Moves 6.Unknowns I. Álvares, Luis Otávio. II. Bogorny, Vânia. III. Título.

ACKNOWLEDGMENTS

A special thanks goes to my advisor, Luis Otavio Alvares, who helped me to grow both as a researcher and as a person. His patience in keep me with the feet on the ground while I was looking at the sky was very important during the master period.

I would like to thank Vania Bogorny, who always demonstrated an incredible motivation and passion in her work and inspired me to reach the same.

Thanks also to my colleagues of the laboratory 232, Fillipo Perotto, who always was ready to a discussion or interchanging ideas. Ivan Medeiros, a rare person who knows a lot about everything, my theory is that he wants to negate the bad fame of the people from Bahia, and I think he is going to do it!

Family members had an important role, my mother Maria do Carmo Tietbohl and my father Paulo José Palma, they hold my position as a master student, which is not an easy task. An expressive part of this degree goes to them. My sister, Denise, for her actions and attitudes. She teached me that sometimes it is easier to act than to think.

Other people that were always present in this period, my friends Fabiane Levemfous and Marcelo Claro Zembrusky. They are examples of people who are always ready to attenuate the stressing period of life. My girlfriend, Polliana Zocche, who tried to awake my creative side in the first year of the master in order to get some good ideas.

I could not forget to thank the Instituto de Informática da UFRGS, the administrative sector and the collegiate sector. They are always helping the students both in the administrative questions or scientific ones.

Additionally, I would like to thank CNPQ for supporting me during my master studies.

“Intuitions without concepts are blind;
Concepts without content are empty.“
— IMMANUEL KANT

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
CDF	Cumulative Distribution Function
CHREST	Chunk Hierarchy and REtrieval STructures
DBMS	Data Base Management System
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
EM	Expectation Maximization
EPAM	Elementary Perceiver And Memorizer
GDPM	Geographic Data Preprocessing Module
GID	Geometric Identification
GPS	Global Position System
MOD	Moving Objects-Database
NNC	Nearest Neighbor Clustering
STPM	Semantic Trajectory Preprocessing Module
SML	Spatial Mining Language
SRID	Spatial Reference Identification
SRS	Spatial Reference System
ST-DBSCAN	Spatial-Temporal DBSCAN
TID	Trajectory Identification

LIST OF FIGURES

ABSTRACT

Because of the large amount of trajectory data produced by mobile devices, there is an increasing need for mechanisms to extract knowledge from this data. Most existing works have focused on the geometric properties of trajectories, but recently emerged the concepts of semantic trajectories, in which the background geographic information is integrated to trajectory sample points. In this new concept, trajectories are observed as a set of *stops* and *moves*, where stops are the most important parts of the trajectory. Stops and moves have been computed by testing the intersection of trajectories with a set of geographic objects given by the user. In this dissertation we present an alternative solution with the capability of finding interesting places that are not expected by the user. The proposed solution is a spatio-temporal clustering method, based on speed, to work with single trajectories. We compare the two different approaches with experiments on real data and show that the computation of stops using the concept of speed can be interesting for several applications.

Keywords: Spatio-temporal clustering, CB-SMoT, stops, moves, unknowns.

Uma Abordagem Baseada em Clusterização para a Descoberta de Lugares de Interesse em Trajetórias

RESUMO

Por causa da grande quantidade de dados de trajetórias producidos por dispositivos móveis, existe um aumento crescente das necessidades de mecanismos para extrair conhecimento a partir desses dados. A maioria dos trabalhos existentes focam nas propriedades geometricas das trajetorias, mas recentemente surgiu o conceito de trajetórias semânticas, nas quais a informação da geografia por baixo da trajetória é integrada aos pontos da trajetória. Nesse novo conceito, trajetórias são observadas como um conjunto de *stops* e *moves*, onde *stops* são as partes mais importantes da trajetória. Os *stops* e *moves* são computados pela intersecção das trajetórias com o conjunto de objetos geográficos dados pelo usuário. Nessa dissertação será apresentada uma solução alternativa a descoberta de *stops*, com a capacidade de achar lugares de interesse que não são esperados pelo usuário. A solução proposta é um método de clusterização espaço-temporal, baseado na velocidade, para ser aplicado em uma trajetória. Foram comparadas duas abordagens diferentes com experimentos baseados em dados reais e mostrado que a computação de stops usando o conceito de velocidade pode ser interessante para várias applicações.

Palavras Chaves: clusterização espaço-temporal, CB-SMoT, stops, moves, unknowns.

1 INTRODUCTION

Thanks to current sensors and GPS technologies, large scale capture of the evolving position of individual mobile objects has become technically and economically feasible. This opens new perspectives for a large number of applications (from e.g. transportation and logistics to ecology and anthropology) built on the knowledge of movements of objects (SPACCAPIETRA et al., 2008). The data provided by these technologies are *raw data* and it becomes difficult to work in a better level of understanding. Usually these data are made up of simple spatio-temporal points, in other words, the points have a spatial location and a time associated. When we sort these simple points by its ascending time we have intuitively a trajectory. Examples of possible trajectories could be the monitoring of wild animals, birds, people, automobiles, airplanes, ships, a soccer player, etc.

Trajectories can be unidimensional, represented as a set of points, (e.g. individual people, birds) or multi-dimensional, represented by a set of polygons or complex objects (e.g. hurricanes, tsunamis, etc). In this dissertation we consider trajectories as a list of points. For details about monitoring of two or more dimensional objects we suggest the reading of Güting's work (GÜTING; SCHNEIDER, 2005).

Usually, raw data are stored by companies only for operational purpose and are not used to extract useful information for decision-making. These raw data are normally represented as a set of single spatio-temporal points, which is intuitively known as trajectory. Usually the points are in the form *(x,y,t)* or *(x,y,z,t)*, where *x, y* and *z* are the spatial dimension and *t* is the temporal one (KUIJPERS; OTHMAN, 2007). Examples of companies that keep this kind of data are: insurance companies, vehicle tracker companies and mobile carriers. Mobile carriers, despite not using GPS devices in order to obtain the raw data, have a similar log of cell phones, like for instance, when a mobile changes from one cell (range of a cellular antenna) to another. Logs keep records about which cellular antenna each cell phone was associated at each time moment, and it might be used in a similar way for the decision-making process.

When raw trajectory data are used in order to extract additional information, as occurs in recent trajectory researches, only the physical properties (spatial and temporal dimensions) use to be focused on (LEE; HAN; WHANG, 2007; GÜTING et al., 2000; GÜTING; SCHNEIDER, 2005; LAUBE; IMFELD; WEIBEL, 2005; WOLFSON et al., 1998). These approaches, which work with raw trajectory data may miss interesting information, since the background geographic information of the trajectories is not considered. If we relate the trajectory's background geography, we are able to discover useful information associated with the trajectory, like for instance the places visited by the trajectory or even the transition between those places. In Figure 1.1 it is possible to see the same set of trajectories considering the background geography and without considering it. So, intuitively we may check that if we have a geographic background context we have more

options in order to aggregate more meaningful information than just consider the trajectory as an isolated object in the space.

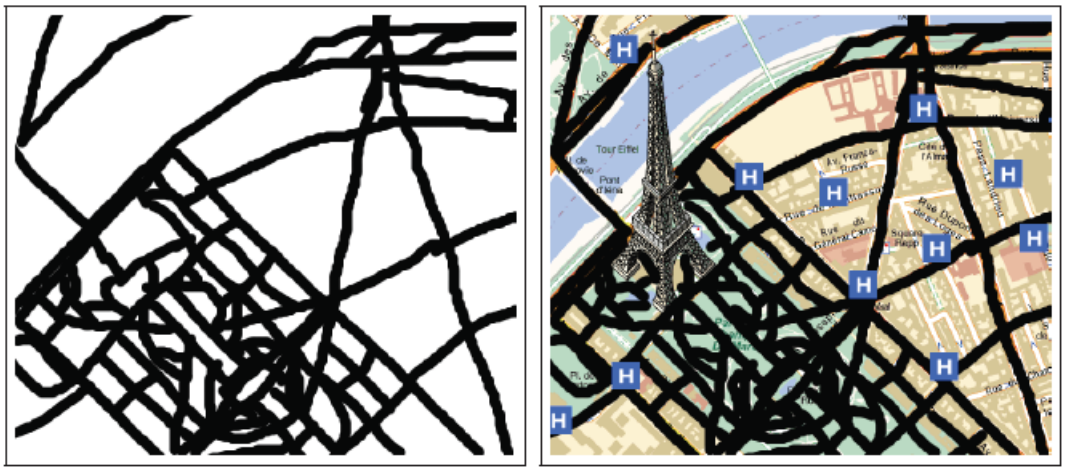

Figure 1.1: The same set of trajectories being considered with and without geographic background (ALVARES et al., 2007)

By associating the background geography it is possible to work in a higher abstraction level, a semantic one, closer to the human understanding. This recent approach was introduced by Spaccapietra (SPACCAPIETRA et al., 2008). In that work he proposed two new concepts, *stops* and *moves*, used to conceptually describe a trajectory. A *stop* represents an important part of the trajectory and a *move* represents transitions between stops.

Spaccapietra in (SPACCAPIETRA et al., 2008) has not defined the criteria to create *stops*, he has just concepted that a stop is a set of consecutive trajectory points. A recent method, called SMoT (ALVARES et al., 2007) (*Stops and Moves of Trajectories*), uses as criteria the intersection of trajectory points with a same place of interest for a given amount of time in order to generate stops. However, in some applications this approach could not be appropriate, since there are some events that are not related to any fixed place, like a traffic jam. Therefore, we propose, in this dissertation, another method, called CB-SMoT (Clustering-Based Stops and Moves of Trajectories), which uses another criterion in order to assign stops: *the speed of the trajectory*. This approach is suitable to identify the cases where the low speed is the main indicator of an interesting event, like in traffic jam.

In addition to the speed criterion, CB-SMoT also uses the background geography in order to aggregate geographic information, similarly to SMoT. Thereupon, CB-SMoT was developed to be used where the speed is a factor as important as the geographic aspects in the background geography. Differently from SMoT, which uses geographic aspects to give semantics to trajectories, CB-SMoT uses speed, which is a spatio-temporal aspect that is not related only to spatial places in the geography.

In order to evaluate the method CB-SMoT we developed a prototype, which is an extension of Weka (WITTEN; FRANK, 2005), which is a well known software developed by University of Waikato and is used for several data mining purposes, like clustering. This extension was originally developed in order to work with geographic data (BOGORNY et al., 2006), and in this dissertation we extended it further for trajectories. The prototype implements both SMoT and CB-SMoT algorithms.

This dissertation is organized in the following way: the next chapter presents the main related works, subdivided in clustering works and trajectories works; Chapter 3 presents

the proposed method to find important points of trajectories. Chapter 4 presents the prototype implemented in order to validate the proposed method and how it was incorporated in Weka. Chapter 5 shows the experiments performed in order to evaluate the method CB-SMoT. For last, the conclusion presents a briefly summary about this dissertation, the main contributions, and some directions for future works.

2 BACKGROUND AND BASIC CONCEPTS

This chapter introduces some works about both trajectory samples and semantic trajectories, and presents the main clustering methods for trajectories.

2.1 Trajectories

From the users' viewpoint, the concept of trajectory is rooted in the evolving position of some object traveling in some space during a given time interval. Thus, a trajectory is by definition a spatio-temporal concept. The concept of traveling object implies that its movement is intended to fulfill a meaningful goal that requires traveling from one place to another. Traveling for achieving a goal takes a finite amount of time (and covers some distance in space), therefore trajectories are inherently defined by a time interval (SPACCAPIETRA et al., 2008). A continuous trajectory can be formally defined as:

Definition 1. *Continuous Trajectory*. A continuous trajectory *T* is the graph of mapping $I \subseteq \Re \rightarrow \Re^2 : t \rightarrow \alpha(t) = (\alpha_x(t), \alpha_y(t))$, i.e., $T = \{(t, \alpha_x(t), \alpha_y(t)) \in \Re \rightarrow \Re^2 | t \in I\}$, where *I* is named *time domain* of *T*.

In other words, a continuous trajectory is a tuple (x,y,t) where *t* is the time and (x,y) are the spatial coordinates associated, where *t* is a strictly positive natural number. However, as important as delimiting the time interval of a trajectory is discretizing it, since it is needed to restrict it to a finite list of points in order to work computationally. So, in the following definition, a trajectory is limited to a discrete number of ordered time-space points:

Definition 2. *Trajectory*. It is a list $\{(t_0, x_0, y_0), (t_1, x_1, y_1), (t_2, x_2, y_2), ..., (t_N, x_N, y_N)\}$, with $t_i, x_i, y_i \in \Re$ for $i = 0, 1, 2, ..., N$ and $t_0 < t_1 < t_2 < ... < t_N$, where t_0 is the instant when the object starts the travel and t_N is the instant when the travel terminates.

In definition 1, the trajectory is seen as a continuous list of points in the continuous time dimension, and in definition 2, the trajectory is discretized.

Most existing works covering trajectories of moving objects only the spatial and temporal dimensions are considered. Some focus on spatial properties, others on the temporal ones, and only a few merge space and time. In general, the trajectory is as an isolated ob-

ject, associated with no other spatial entity, unless, in some cases, with other trajectories.

Güting (GÜTING et al., 2000; GÜTING; SCHNEIDER, 2005; BECKER; JENSEN; SU, 2007) presents a full approach over modeling moving object databases (MODs). A MOD is a database developed to store and provide useful operations to manipulate trajectories, supporting both geometric and temporal properties. These works cover the whole development of a MOD, from query languages, passing through generic spatio-temporal data types and operations for these data. As MODs store moving objects and/or moving regions, they have specific data types in order to represent such elements, like moving *points, multipoints, lines or regions*. Some MODs use vectors in order to represent the speed of an object in order to decrease the number of stored points.

Another important element in DBMS query languages are the operators. A MOD with support for many different kinds of spatial and temporal operators use to have a richer DBMS query language, since more options are available to perform queries. Examples of common operators are *intersection*, which tests if some object intersects another; *distance* calculates the euclidean distance between two elements; or semantic operators like *may* and *must*, which indicate uncertainly of the location of a point in a polygon (WOLFSON et al., 1998) or either in a cube in the (x,y,t) coordinates system.

New operators used in existing query languages are necessary to obtain information about moving objects, like for instance, *When and where was the spread of fires larger than 500* km^2*?* or *How many taxis are at most 10 minutes from John's house?*. Additionally, some uncertainty query mechanisms can be provided, since the locations of moving objects may not be exact.

Wolfson (WOLFSON et al., 1998) tries to approximate the location of a moving element storing the speed vector. So, in order to avoid frequent updates in the object position, a vector that simulates the movement of the object is stored. As update the vector is less constant than update the own object coordinates, that approach avoids a bottleneck in the database. However, it is a tradeoff between the object precision location and the database updates. As his option was to reduce the database updates, precision is lost in the object's coordinates, and the queries are not precise.

Brakatsoulas (BRAKATSOULAS; PFOSER; TRYFONA, 2004) combines trajectory data with spatio-temporal information. The geographic place modeled is the city of Athens. In their work they present a spatial mining language (SML) to answer queries and provide useful information about states of their MOD.

In Brakatsoulas (BRAKATSOULAS; PFOSER; TRYFONA, 2004) model, each trajectory is seen as a line that goes up the temporal axis in a 3-dimensional space-time coordinate system. Additionally, the queries represent a polyedron in that spatio-temporal axis, and depending on how the trajectories cross this region, they may or not be in the query answer, as shown in Figure 2.1 (a). In the same figure, in (b), are presented some basic spatial relations found between trajectories.

Another approach about trajectories is seen at Guting (GüTING; ALMEIDA; DING, 2004), where trajectories are seen over networks. As most routes scenarios can be modeled by a graph (edges and nodes), a network was a useful way to see a trajectory. The goal of that work is to provide a comprehensive data model and query language for moving objects in networks, supporting the description and querying of complete histories of movement. The main characteristics are:

- The model has a explicit concept of network embedded in space.
- A moving object is described relative to the network rather than the embedding

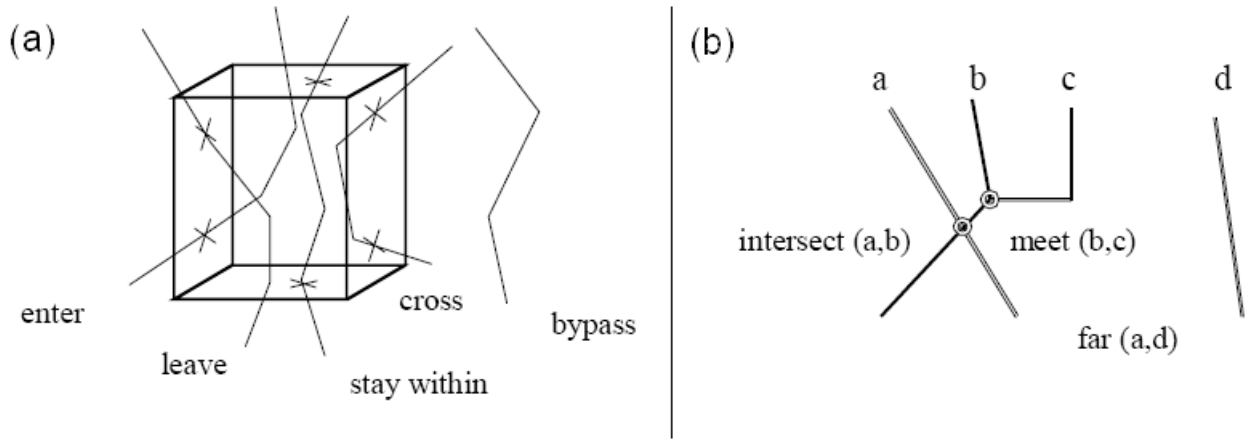

Figure 2.1: In (a) the relations between trajectories and the environment; and (b) relations between trajectories (BRAKATSOULAS; PFOSER; TRYFONA, 2004)

space.

- The user extends the network information using the facilities of the DBMS.
- The model supports static or moving objects only in relation to the network model.
- There exists a framework to represent and query objects moving freely in the space.

The use of a network as a model can be interesting because most trajectories happen in a well defined place, like street, fly routes, etc. In this dissertation we do not focus on network models because it is not generic enough to enclose all possible scenarios of trajectories. For instance, in the scenario of bird migration, we dont't have routes well defined, as well as in a hurricane trajectory and other climatic scenarios.

2.1.1 Conceptual Trajectories

Some definitions about trajectories over both physical and semantic perspective are presented in this section.

In Mark (MARK et al., 1999) trajectories are related to a large period of life (lifelines) and are associated to individuals' locations at regular or irregular temporal intervals. The objective was to find spatial clusters in the past or to determine past environmental exposures, like, for example, causes of diseases and exposure to toxic substances. The period of life ranges from years to decades, since chronic diseases use to have as cause an extended exposure to an injurious antecedent. Figure2.2 shows three lifelines for several decades.

Thériault (THERIAULT et al., 2002) followed a similar way building a spatio-temporal database model for handling lifelines allowing statistic analysis of any pre-defined event. Each lifeline is related to some aspect of the individual life, for instance, professional, residential, household, and so on, and it is divided in episodes and events, as can be seen in Figure 2.3. So, it is possible to find out evolution (or decision) patterns. For example, if we discover that 70% of people which lived in a industrial region for at least 5 years, would have, in a posterior moment of their life, some kind of cancer.

These works started giving semantics to trajectories some years ago. However, a more recent approach came out, as will be explained in the next section.

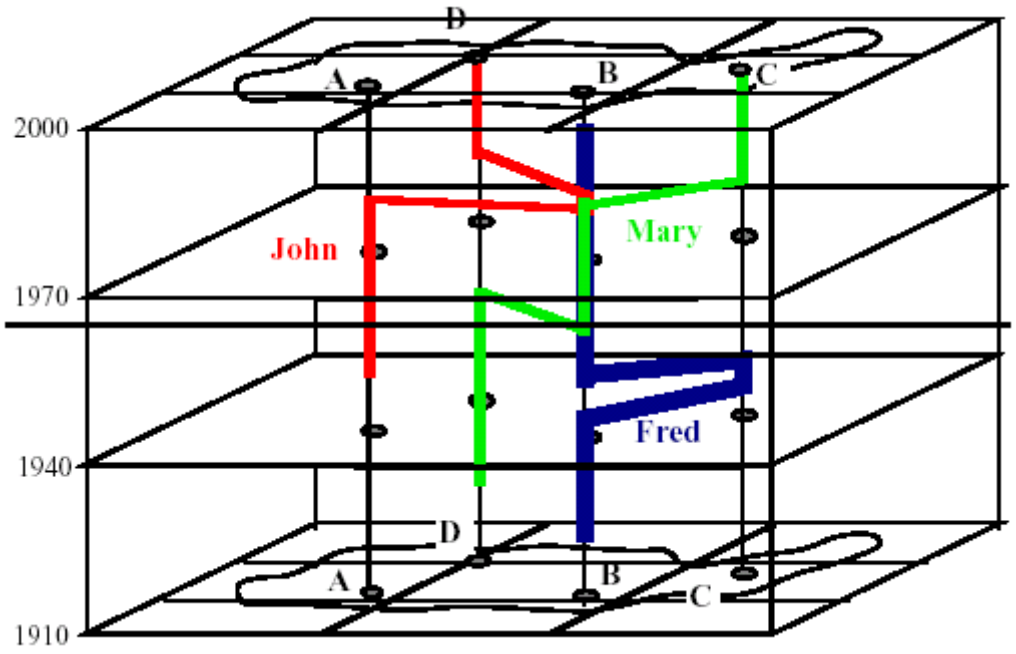

Figure 2.2: Lifelines of three people showing the cities where they lived (MARK et al., 1999)

2.1.2 Stops and Moves

A recent approach was introduced by Spaccapietra in 2007 (SPACCAPIETRA et al., 2007, 2008). This approach defines a conceptual model for trajectories. A specific concern is to model trajectories with semantic annotations, allowing users to define semantic data to specific parts of the trajectory. This approach can be used by many applications which need a more structured recording of the movement. (i.e. a temporal sequence of journeys). These countable journeys are referred as trajectories, and denote the involved object as the traveling object. Applications may use these trajectories and analyze them, for example to derive mobility patterns to be used in some decision making process like, for instance, knowledge for optimizing traffic management, bird trajectories, control the implementation of transportation logistics, etc.

The exact information aggregated to the trajectory is related to the context of the application, but essentially the trajectory is partitioned in smaller pieces, called *stops* and *moves*, and the information added gives a more meaningful understanding to stops and/or moves. For instance, in a study about migrations of birds, we could add the following information to stops:

- the kind of stop: a briefly or a longer one (resting and eating);
- the time it happened;
- the weather condition.

So, we could study the reasons why these specific birds stop. On the other hand, in the moves could be considered:

- the mean speed;

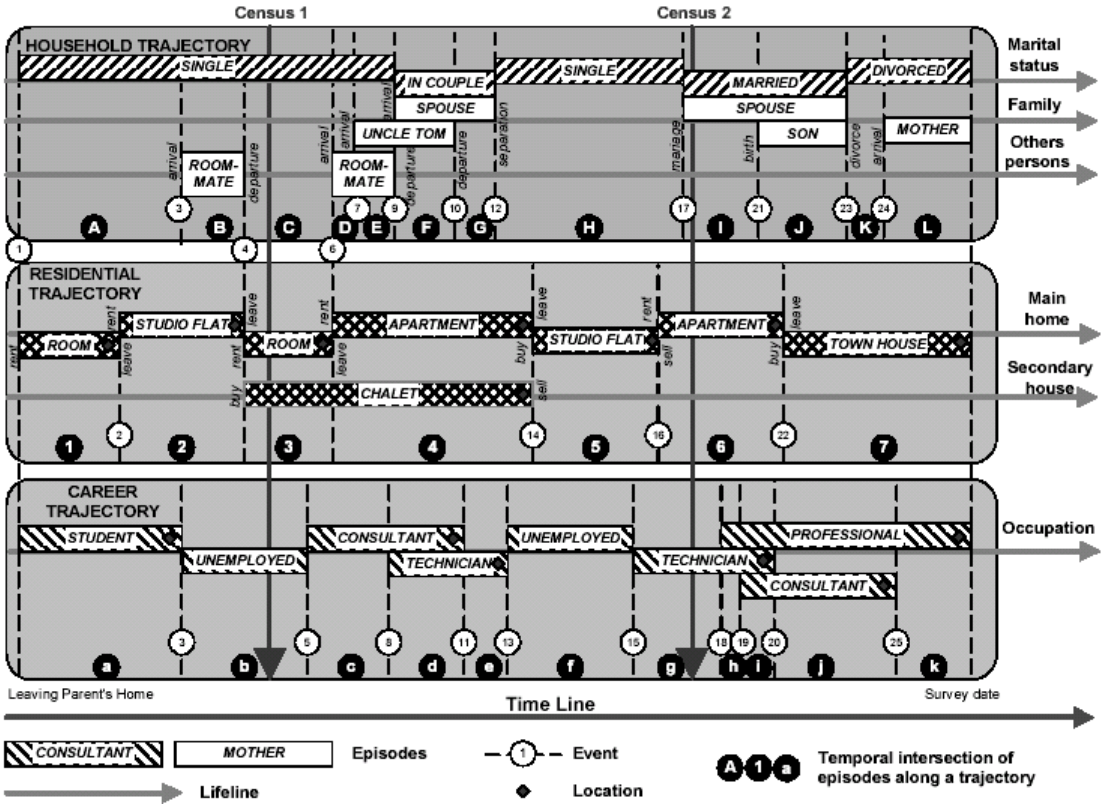

Figure 2.3: Different kinds of trajectory of an individual history (THERIAULT et al., 2002)

- the altitude at which the birds are flying;
- the weather condition.

Similarly, it would be possible to discover the causes why the birds are flying higher or faster, or any other characteristics, which will be dependent only on the information recorded in that basic piece of trajectory.

Spaccapietra (SPACCAPIETRA et al., 2008) has not specified what information can be aggregated to a stop/move, however he defined some characteristics about them:

Stop: A stop is a trajectory relevant time interval, such that:

- the user has explicitly defined this part of the trajectory to represent a stop;
- the temporal extent is a non-empty time interval;
- the traveling object does not move, as far as the application view of this trajectory is concerned;
- all stops in a same trajectory are temporally disjoint, i.e. the temporal extents of two stops are always disjoint.

Move: A move is a part of the trajectory, such that:

- The part is delimited by two extremities that represent either two consecutive stops, or t_{begin} and the first stop, or the last stop and t_{end}, or $[t_{begin}, t_{end}]$ (case when the trajectory has no stops).

- The temporal extent $[t_{begin}, t_{end}]$ is a non-empty time interval.

- The spatial range of the trajectory for the $[t_{begin}, t_{end}]$ interval is the spatio-temporal line (not a point) defined by the trajectory function.

Where t_{begin} is the initial point of the trajectory and t_{end} is the final one.

The concepts of stops and moves are key ideas used in this dissertation when the trajectory becomes viewed from different points of view, not looking at physical properties, but at semantic properties.

2.1.3 Enriching Trajectories with Semantic Geographical Information

Spaccapietra (SPACCAPIETRA et al., 2008) has not defined any process to automatically assign semantic information, he has just exemplified some scenarios where his model could be used. This is because his research group focuses on conceptual modeling, and they don't care on how stops are computed. In Alvares (ALVARES et al., 2007), a data preprocessing model and an algorithm is presented to add semantic information to trajectories in order to facilitate trajectory data analysis in different application domains. This work defines mathematically some concepts in order to instantiate the more abstract work of Spaccapietra.

The first important concept is the *Candidate Stop*, which is defined as follows:

Definition 3. *Candidate Stop*. A candidate stop *C* is a tuple (R_C, Δ_C), where R_C is a (topologically closed) polygon or polyline in R^2 and Δ_C is a strictly positive real number. The set R_C is called the geometry of the candidate stop and Δ_C is called its minimum duration.

Definition 3 indicates what are the objects that may become a stop. Thus, it says that a candidate stop has two properties, one spatial and one temporal. The spatial property, R_C, represents the location and shape of the place (e.g. polygon or lines). For instance, a park would have a polygon as its R_C and a street would have a line in this attribute. The temporal aspect indicates the amount of time the object spends in that place. For example, in a large museum, for instance, it is expected that people spend at least one hour in it, so the museum parameter would be 1 hour. Hence, the temporal parameter avoids assigning the candidate stop as an important place for objects that were only crossing the place.

Given a set of candidate stops and a set of trajectories it is possible to set aside as stops those candidate stops that match the conditions. However, we may not be interested in every place the trajectories visited, so it is interesting to limit the set of candidate stops used in the search. This restriction is usually related to the application. For instance, in a tourism application it is not interesting to characterize the stops of tourists at gas stations. Therefore, gas stations will not be considered as candidate stops. In a similar way, in a traffic application, the set of candidate stops could be limited only to streets (and avenues), traffic lights, viaducts, bridges, crossings, etc. For this reason, another restrictive definition is needed, called application (ALVARES et al., 2007) (see Figure 2.4):

Definition 4. *Application*. An application is a finite set $A = \{C_N = (R_{cN}, \Delta_{cN})\}$ of candidate stops such that each element has non-overlapping geometries, $R_{c1}, R_{c2}, ..., R_{cN}$, with each other one.

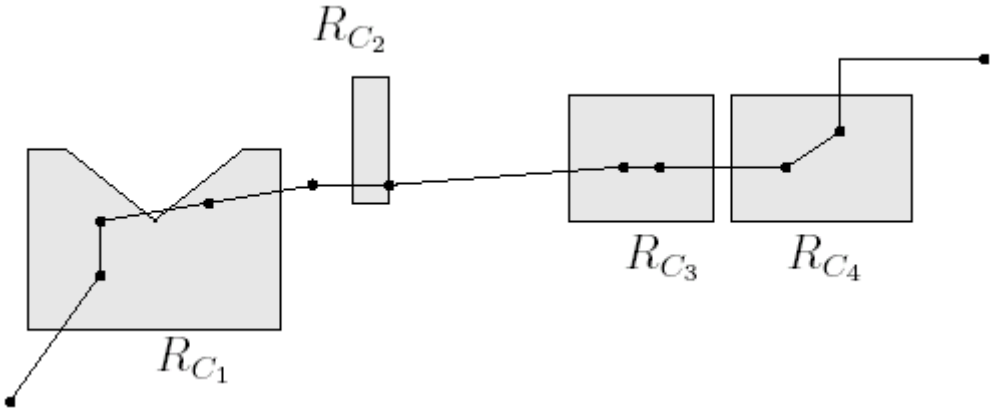

Figure 2.4: A trajectory and an example of application with four candidate stops (ALVARES et al., 2007)

Given an application, it is possible to identify the stops with respect to the set of trajectories. Alvares (ALVARES et al., 2007) defines a stop (with respect to a trajectory) based on the amount of time of the continuous intersection of the trajectory with a candidate stop, formally, given as:

Definition 5. *Stop*. Let *T* be a trajectory with time domain *I* and let $A = (\{C_1 = (R_{C_1}, \Delta_{C_1}), ..., C_N = (R_{C_N}, \Delta\_C_N)\})$ be an application. Let $I_0, I_1, ..., I_n$ be the temporally-ordered decomposition of *I* into time intervals where *T* is intersected by candidate stops with respect to *A*. A stop of *T* with respect to *A* is a contiguous sub-trajectory of *T* over a maximal concatenation of some $t_i, t_{i+1}, ..., t_{i+\ell}$ for which there is a (R_{C_k}, Δ_{C_k}) in *A* such that:

- $< (x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1}, ..., (x_{i+\ell}, y_{i+\ell}, t_{i+\ell})) >$ intersects R_{C_k}.
- $|t_{i+\ell} - t_i| \geq \Delta_{C_k}$

In other words, the maximum continuous intersection time of a trajectory with a candidate stop must be greater or equal to the candidate stop's Δ_C, in order to a candidate become actually a stop with respect to that trajectory, as shown in Figure 2.5.

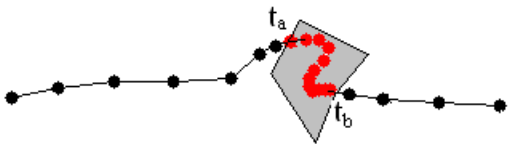

Figure 2.5: Intersecting points of the trajectory must intercept the candidate stop polygon (in gray) for a time period $t_b - t_a \geq \Delta_C$

The move definition used in Alvares (ALVARES et al., 2007) follows Spaccapietra (SPACCAPIETRA et al., 2008). Figure 2.4 illustrates these concepts. In this example, there are four candidate stops with geometries $R_{C_1}, R_{C_2}, R_{C_3}$ and R_{C_4}. Let us imagine that the spatial projection of the trajectory *T* is run through from left to right and the

black points are the time points of T. First, T is outside any candidate stop, so we start with a move. Then *T* enters R_{C_1} for three time points. if the duration of staying inside R_{C_1} is long enough, (R_{C_1}, t_1, t_3) is the first stop of *T*. Next, *T* enters R_{C_2}, but it does not generate any time interval, so this is not a stop. We therefore have a move until *T* enters R_{C_3} and, after, R_{C_4}, which could generate or not, according the intersection time period. The trajectory *T* ends with a move. Figure 2.6 illustrates the four possible cases of moves in light gray. In (1) the trajectory begins with a *move*. In (2) the *move* is between two *stops*. In (3) the trajectory ends with a *move*. For last, in (4) there is no *stop* and the whole trajectory is a *move*. That figure shows each case isolated, but they can merge among them, like in (2), where in fact the trajectory begins with a *move*, has an intermediate *move* (in light gray), and ends with a *move*.

Figure 2.7 represents a specific case of situation 2, where there are two consecutive stops, but no intermediate trajectory points between them. So the move will be made up of the two extreme intersection points of the *stops*. In this case these points will be both *move* points as *stop* points. In a general way, the first stop point and the last stop point of a candidate stop will be usually move points also. They are *stop* points since they are inside a candidate stop and they are also *move* points because a *move* needs start/end in a stop in order to be delimited.

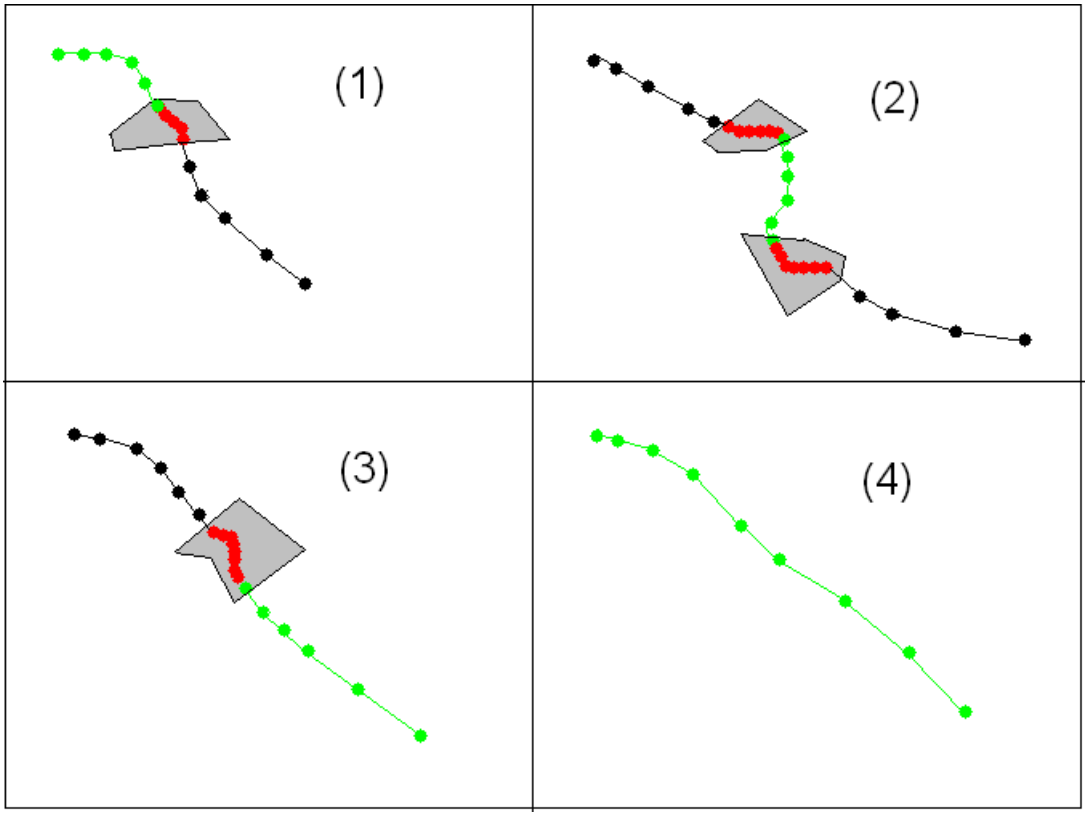

Figure 2.6: Candidate stops as polygons, stops in dark gray inside the polygons and the light gray represents the four possible cases of moves

The main purpose of using stops and moves is that it significantly facilitates trajectory queries and analysis (ALVARES et al., 2007). Instead of working in a lower level of comprehension, the semantic approach lets the structure of queries easier to the human understanding, since it acts in a more abstract level and it provides a more powerful query tool. Besides, the data mining and KDD can take advantage of it due to the more pow-

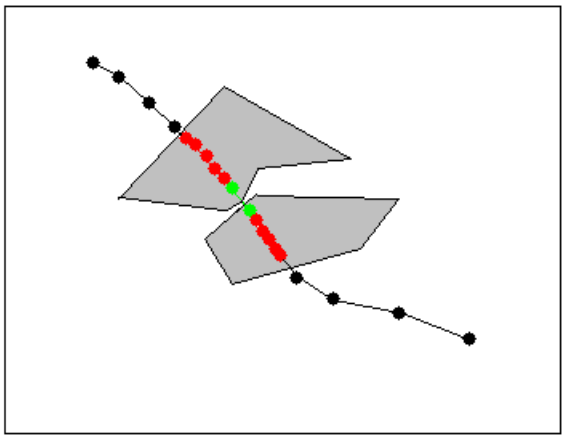

Figure 2.7: Between two consecutive stops there will be a move of two points.

erful semantic representation. Thus, those applications may take advantage of this new approach in order to amplify the quality of semantic data being mined, as we describe in (ALVARES et al., 2007).

The algorithm proposed by Alvares (ALVARES et al., 2007) in order to identify the stops and moves of a trajectory with respect to an application is called SMoT, which stands for *Stops and Moves of Trajectories*. The algorithm looks for continuous subtrajectories that intersect the same candidate stop (with respect to an application) for its minimum time duration.

The algorithm SMoT performs a loop in the trajectory points and tests if each point intersects a candidate stop in the set of candidate stops (application). Whether there is a continuous intersection of points with the same candidate stop then that intersection duration is calculated and matched against the candidate minimum time duration. If this intersection lasts at least for that candidate minimum time duration, this part of trajectory will be a stop. This process is repeated for every point and at the end of the process we will have the list of stops. The parts of a trajectory that are not stops are moves. A move is defined by its start and end stops, and by the subtrajectory between these two consecutive stops[1].

There is an optimization in SMoT algorithm. It is possible to test if a following point P_f (calculated based on the candidate stop's minimum time) intersects the same candidate stop of the current point (P_g). In the positive case, it is necessary to go back and test each intermediate point between them, since it is needed to guarantee that the intersection is continuous. Otherwise, the loop is advanced to P_f, because the continuous intersection will fail for that point and it implies no stop would be created for such points.

SMoT's complexity in the worst case occurs when:

1. **The optimization is not useful:** Each intersecting point has a following intersecting point associated to the same candidate stop, but the intersection is not continuous.

2. **There are no stops:** In this case, each trajectory point will be matched with every candidate stop.

[1]If the trajectory starts or ends with a move, those moves will not have a start stop or end stop respectively

In that case, suppose a trajectory with N points and the set of C candidate stops arranged in a R-Tree structure. As the access to the candidate stop is $O(LogC)$, the final complexity in the worst case will be $O(NLogC)$, when it is necessary to test each point with each candidate stop[2].

2.2 Clustering Methods

A cluster is a group of data that share similar properties previously chosen. Usually each cluster is a subset of the whole dataset. The term 'clustering' is used in several research communities to describe methods for grouping unlabeled data (JAIN; MURTY; FLYNN, 1999), figure 2.8 shows a taxonomy according these authors.

In Dubes and Jain (DUBES; JAIN, 1976), a set of admissibility criteria defined by Fisher and Van Ness (FISHER; NESS, 1971) are used to compare clustering algorithms. These admissibility criteria are based on: (1) the manner in which clusters are formed, (2) the structure of the data, and (3) sensitivity of the clustering technique to change those do not affect the structure of the data. There are several approaches to clustering data. The most comom are hierarchical, partitional and density-based.

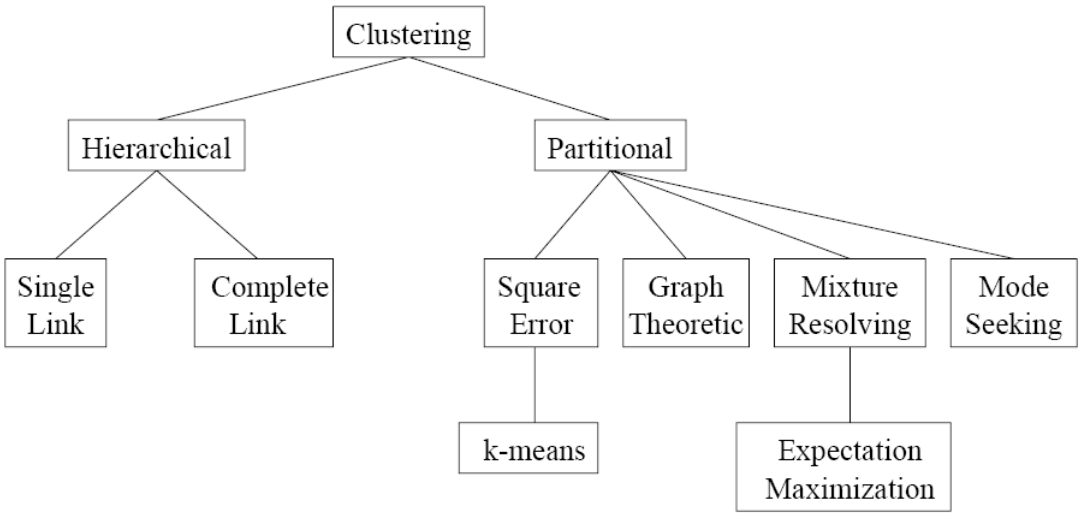

Figure 2.8: Taxonomy of clustering approaches (JAIN; MURTY; FLYNN, 1999)

The hierarchical clustering arranges the clusters into a hierarchical tree, where each sub-level is a specialization of the upper one. In this kind of clustering, the hierarchical tree may be constructed in two ways: bottom-up (agglomerative) or top-down (divisive). The first begins with each pattern in a distinct (singleton) and successively merges clusters together until a stopping criterion is satisfied. A divisive method begins with all patterns in a single cluster and performs splitting until a stopping criterion is met. Most hierarchical clustering algorithms are variants of the single-link (SNEATH; SOKAL, 1973), complete-link (KING, 1967), and minimum-variance (WARD, 1963; MURTAGH, 1983) algorithms. The single-link and complete-link algorithms are the most popular. These two algorithms differ in the way they characterize the similarity between a pair of clusters. In

[2]A R-Tree acts like a filter too, eliminating those candidate stops that dont't belongs to the region associated with the point. Hence the amount of candidate stops tested actually with each point will be a subset of the whole set, which is much smaller than the total amount of candidate stops

the single-link the similarity between a pair of clusters is the shortest distance between all pair of patterns (one from each cluster), on the other hand the complete-link is based on the maximum distance between them (JAIN; MURTY; FLYNN, 1999).

The classification process of events, facts and observations without a teacher monitoring (non-supervised learning) is called *conceptual clustering*, and it was defined as a machine learning task in the eighties by Michalski (FISHER, 1987). Conceptual clustering uses to be constructed in a tree like structure, since the knowledge classification is normally hierarchical. Additionally, the clustering building has an incremental aspect, in other words, each instance of data is presented one by one, so that the whole dataset is not known a priori by the method.

Some concepts used in psychology, for instance the *chunking* idea (FERNAND GOBET PETER C. R. LANE; PINE, 2001), which is a unification mechanism of information inherent in the human psychology, have influenced the initial works in conceptual clustering, like Feigenbaum work (FEIGENBAUM EDWARD A., 1984, 1962; FEIGENBAUM, 1963) and the model CHREST (*Chunk Hierarchy and REtrieval STructures*) (FERNAND GOBET PETER C. R. LANE; PINE, 2001). Later, another model was created inspired in the Feigenbaum's model: UNIMEM (LEBOWITZ, 1987), which was created by Lebowitz. UNIMEN is able to treat also numeric attributes in relation to Feigenbaum's model (EPAM), which is not able to do, as well as the symbolic ones. While EPAM was used in verbal learning experiments, UNIMEM was used in more complex tasks, such as natural language understanding and inference (GENNARI; LANGLEY; FISHER, 1989). Besides, it is able to keep a pattern in more than one category (cluster).

An important hierarchical clustering algorithm has been developed by Fisher, named COBWEB (FISHER, 1987). Despite it has not been created in order to be a psychological model, it influenced the cognitive psychology field. Fisher's model makes use of probability in order to assign the importance of each category. The COBWEB's function tries to promote the inference potential by maximizing the intra-class similarity and the inter-class differences. So, we have a trade off between *predictability* and *predictiveness*.

Other related hierarchical clustering methods are: CLASSIT (GENNARI; LANGLEY; FISHER, 1989), which is able to deal with numerical attributes; ARACHNE (IBA; LANGLEY, 2001), which uses alternative heuristics in order to build the hierarchical tree; LABYRINTH (THOMPSON; LANGLEY, 1991) that tries to integrate the best features of the previous models; and OXBOW (IBA, 1991) that allows clustering data with temporal domain and it is used to learn moving related concepts.

Partitional clustering algorithms obtain a single partition of the data instead of a single structure, like the tree structure produced by hierarchical techniques. In other words they start with the whole data set and create a partition in this dataset, each one representing a cluster. A problem found in this approach is that the number of output clusters must be known a priori. Usually the clusters are produced by optimizing a function defined either locally or globally. Combinatorial search of the whole set for an optimum value of a criterion is prohibitive.

The most common and intuitive criterion used is the *squared error*, which is to minimize the feature distances between the elements of a cluster and its centroid. The function that represents the quality of the cluster is the following (JAIN; MURTY; FLYNN, 1999):

$$e^2(\chi, \zeta) = \sum_{j=1}^{K} \sum_{i=1}^{n_j} |X_i^{(j)} - C_j|^2$$

Where χ is the dataset (with K clusters) and ζ is a clustering of it. $X_i^{(j)}$ is the i^{th}

element belonging to the j^{th} cluster and C_j is the centroid of the j^{th} cluster.

K-means (MACQUEEN, 1967) is the simplest and common algorithm that uses the squared error as criterion. It operates by positioning the centroid pattern in a way that the squared error is less or equal to some threshold. It is an efficient iterative clustering algorithm, but this class of algorithms have an inherent issue, the number of clusters must be known a priori. K-Means uses a parameter *k* representing the number of initial patterns (centroids) which will move among the dataset in direction to the center of the clusters. It iterates through each point, finds the nearest centroid, and assigns the point to that pattern. This iteration is repeated until the error term is deemed small or not decreasing much. The *k* parameter (the number of clusters) must be known a priori and this restriction makes the method inadequate for our proposal.

Another approach is the *graph-theoretic clustering*, whose best-known algorithm is based on the construction of the *minimum spanning tree* (MST) (ZAHN, Jan. 1971) of the data, and then deleting the edges with largest lengths in order to generate clusters, as shown in Figure 2.9.

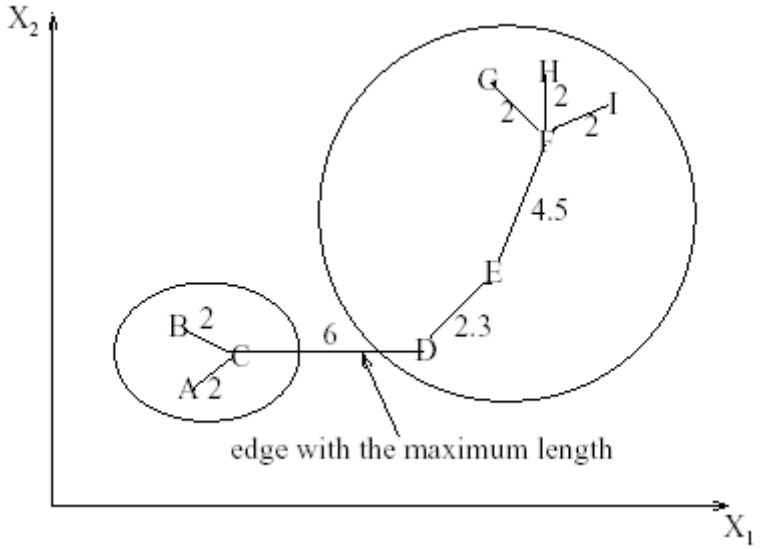

Figure 2.9: Deleting the largest edge in order to form two clusters (JAIN; MURTY; FLYNN, 1999)

A usual way to evaluate the quality of a clustering is to use a distribution function in order to create the group of data (potential clusters). So, the goal of a clustering algorithm is to group the instances of similar value in order to achieve the best value in that distribution criterion function. Those instances, placed (by their scores) in a particular component, would therefore be viewed as belonging to the same cluster. Traditional approaches to this problem involve obtaining (iteratively) a maximum likelihood estimative of the parameter vectors of the component densities (JAIN; DUBES, 1988). The Expectation Maximization (EM) is a general purpose maximum likelihood algorithm (DEMPSTER; LAIRD; RUBIN, 1977) for missing data problems. The EM procedure begins with an initial estimate of the parameter vector and iteratively rescores the patterns against the mixture density produced by the parameter vector. The rescored patterns are then used to update the parameter estimatives (MITCHELL, 1997).

Similarity is a key notion in clustering algorithms. When the similarity criterion becomes the distance among elements we have a *nearest neighbor clustering* (NNC)

method. An iterative procedure was proposed in Lu and Fu (LU; FU, May 1978); it assigns each unlabeled element to the cluster of its nearest labeled neighbor, given that the distance to that labeled neighbor is below a threshold. This process continues until all patterns are labeled or no additional labellings occur (JAIN; MURTY; FLYNN, 1999). When we work with clustering of spatial data, the physical distance uses to be an intuitive criterion in order to generate clusters in NNC methods.

A clustering algorithm has other classifications, according its features. With respect to the precision of an element in the cluster, they can be *hard* or *fuzzy* (also named *soft*). A hard clustering approach labels an element to just one cluster, on the other hand, the fuzzy technique allows an element to be in many clusters with a respective probability. With respect to the full knowledge of the data to be clusterized, they can be *incremental* or *non-incremental*. An incremental method does not known a priori the data set, and the elements to be clusterized are presented as input one by one. In these techniques at each time an element is presented as input, the method rearranges the cluster structure in order to accommodate the new element, so the presentation order is important. A non-incremental approach knows a priori all elements at the beginning of the clustering process, it usually is a harder processing because it requires more memory.

In the next sections will be presented the main methods which influenced this dissertation. Those clustering methods are density-based, in other words, they consider the amount of points in a given region and try to expand the cluster by adding regions with similar density.

2.2.1 DBSCAN

DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) is an important density clustering method. It was developed by Ester (ESTER et al., 1996) in 1996 and it has influenced many other density-based clustering methods. The main advantage of DBSCAN is the capability to find clusters of many different shapes in a spatial dataset. Figure 2.10 shows some datasets where it is able to discover the intuitive visual clusters.

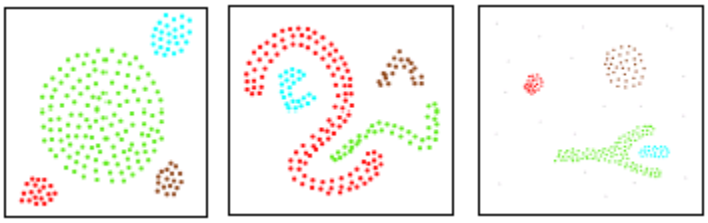

Figure 2.10: Three datasets with its respective clusters (ESTER et al., 1996)

DBSCAN looks for some core cluster and tries to expand it by aggregating the nearest neighbors that met some conditions. In order to understand what these conditions are, it is necessary to present some related concepts. In fact, DBSCAN needs two parameters: *MinPts* and *Eps*. *MinPts* is a density measure that indicates the amount of points needed in a neighborhood of a point in order to assign that point and its neighbors to a cluster. *Eps* is a distance used to delimiting the neighborhood. Formally the neighborhood is defined as follows:

Eps-neighborhood of a point: The Eps-neighborhood of a point p, denoted by $N_{Eps}(p)$ is defined by $N_{Eps}(p) = \{q \in D | dist(p,q) \leq Eps\}$, where D is the whole dataset.

The Eps-neighborhood of p represents all points inside a circle centered in p and with radius Eps. The $dist(p,q)$ is a function that returns the distance between the point p and

the point *q*, considering the Euclidean distance on a 2D surface. A simpler approach could consider as a cluster every point which has a minimum *MinPts* inside its neighborhood, but this approach fails because there are two kinds of points in a cluster: those in the center and those in the border. The latter has less points than the former, as shown in Figure 2.11 (a). To resolve it, they have created some other definitions that build step by step the solution and is illustrated in figure 2.11 (b).

Directly Density Reachable: A point *p* is directly density reachable to a point *q* with respect to Eps, $MinPts$ if:

1. $p \in N_{Eps}(q)$
2. $|N_{Eps}(q)| \geq MinPts$ (Core point condition)

A point which meets the second condition is called *core point*, otherwise it called *border point*.

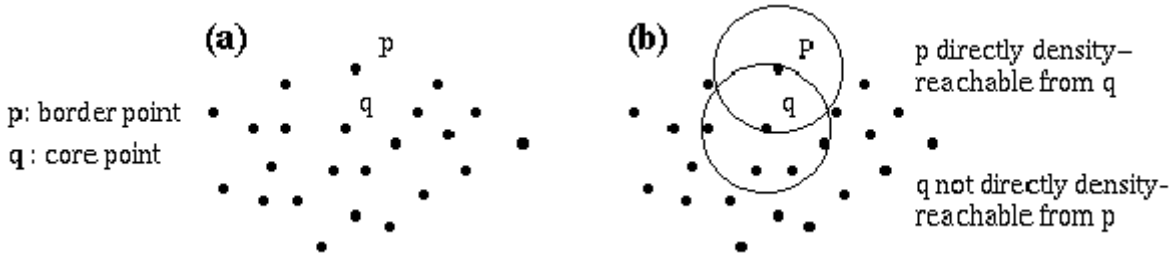

Figure 2.11: (a) Border and core points; (b) Directly Density Reachable illustration (ESTER et al., 1996)

The directly density reachable illustration is symmetric for pairs of core points, but is asymmetric for a pair border/core points. The next definition relates every border point with the core points, since there is a chain of directly density reachable points.

Density Reachable: A point p is density reachable to a point q with respect to Eps, $MinPts$ if there is a chain of points $p_1, p_2, ..., p_n; p_1 = q, p_n = p$ such p_{i+1} is directly density reachable from p_i.

Now it is necessary to relate two border points in the same cluster. Thus the *density connected* definition provides it.

Density Connected: A point p is density connected to a point q with respect to Eps, $MinPts$ if there is a point o such that both p and q are density reachable from o with respect to Eps and $MinPts$.

The density reachable and density connected definitions may be better visualized in Figure 2.12.

With respect to the definitions above, it is possible to define both cluster and noise according DBSCAN:

Cluster: Let D be a database of points. A cluster C, with respect to Eps and $MinPts$, is a non-empty subset of D satisfying the following conditions:

1. $\forall p, q$: if $p \in C$ and q is density reachable from p with respect to Eps and $MinPts$ then $q \in C$. (Maximality)
2. $\forall p, q$: p is density connected to q with respect to Eps and $MinPts$. (Connectivity)

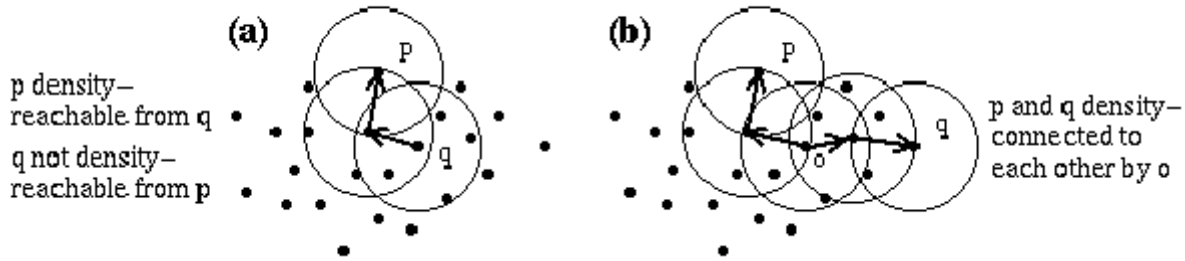

Figure 2.12: (a) Density Reachable definition; (b) Density Connected definition (ESTER et al., 1996)

Noise: Let $C_1, C_2, ..., C_n$ be the clusters of the database D with respect to Eps and $MinPts$. Then is defined as noise the set of points in the database D not belonging to any cluster C_i. Noise = $p \in D | \forall i : p \notin C_i$.

DBSCAN's advantage is that it has a good performance on large spatial databases, outperforming other density-based algorithms like CLARANS (NG; HAN, 1994). DBSCAN's complexity is $O(nlogn)$, considering the data on a R*-Tree (BECKMANN et al., 1990) structure. It requires little knowledge about the spatial domain, since it needs only two parameters in order to be performed: $MinPts$ and Eps. Besides, the authors that proposed DBSCAN proposed a method based on a graphic in order to estimate a good value to the Eps parameter. In this method the $MinPts$ is set to 4 and the Eps is calculated based in the sorted 4-dist graph. The user must look for a valley in the graph, which represents the threshold point to be used as Eps [3].

The main disadvantage of DBSCAN is its difficulty to find clusters of different densities, like those shown in Figure 2.13. That figure shows 4 distinct clusters, where the cluster A has the greatest density among them and the cluster D has the lowest one. The clusters B and C have intermediate densities.

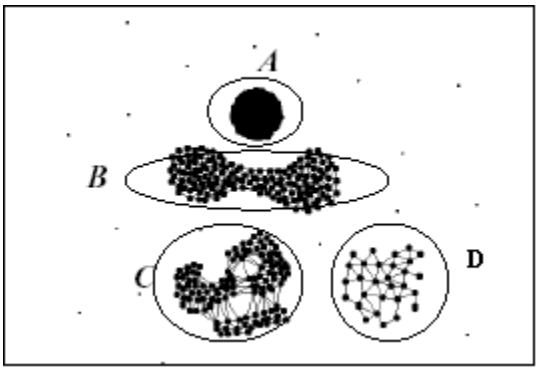

Figure 2.13: Clusters with different granularities (ESTIVILL-CASTRO; LEE, 1999)

2.2.2 DJ-Cluster

DJ-Cluster (ZHOU et al., 2004, 2007) is an algorithm created after DBSCAN. It was projected after some experienced problems with DBSCAN's performance. According to DJ-Cluster creators, for some Eps and $MinPts$, DBSCAN algorithm will generate a

[3] This process is not considered in the DBSCAN complexity

large number of points within its density definition, each of which could be further used to generate its own density-reachable points. In such cases, it will use a lot of memory and slow down considerably.

Where DBSCAN uses the connectivity notion of a clique graph, DJ-Cluster instead uses the concept of connected components. This helps DJ-Cluster to avoid the performance problems mentioned. However, DJ-Cluster is simpler than DBSCAN despite some common definitions, like Eps-neighborhood. Two other definitions are necessary to define a cluster according DJ-Cluster.

Density Joinable: $N_{Eps}(p)$ is density joinable to $N_{Eps}(q)$, denoted as $J(N_{Eps}(q), N_{Eps}(p))$, with respect to Eps and $MinPts$, if there is a point o such that both $N_{Eps}(p)$ and $N_{Eps}(q)$ contain o.

DJ Cluster: The density and join based cluster C is defined as follows: $\forall p \in D, \forall q \in D, \exists N_{Eps}(p), N_{Eps}(q)$ such that $\exists J(N_{Eps}(q), N_{Eps}(p))$.

Figure 2.14 exemplifies both definitions. At Figure 2.14 (a) the neighborhoods of p and q, which met the $MinPts$ condition, share a common point o, so they are merged in a larger single cluster, illustrated in Figure 2.14 (b).

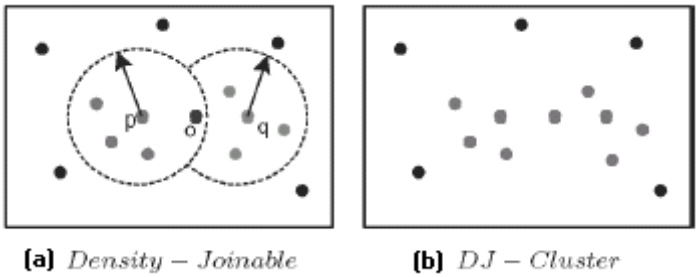

Figure 2.14: (a) Density Joinable definition; (b) DJ-Cluster formed by joining the two neighborhoods in (a) (ZHOU et al., 2007)

DJ-Cluster was used to discover personal gazetteers, and it has been more precise in this task than the k-means clustering method. In this approach the baseline (important) places are specified by the user, and DJ-Cluster checks if these places occur in the trajectories. So, four possibilities may happen:

- Baseline places that were discovered.
- Baseline places that were not discovered.
- Discovered places that are interesting and significant to the person.
- Discovered places that are not interesting to the person.

DJ-Cluster is a very interesting clustering algorithm, but it does not bind automatically the clusters to important geographical places, which are called *personal gazetteers*. In some sense, it is a method to assign semantic information to the clusters, but it needs user's assistance.

The high level algorithm is presented in listing 2.1 (ZHOU et al., 2007).

Listing 2.1: DJ-Cluster Algorithm

```
INPUT:
        S       : Sample dataset of points\\

1:      WHILE there is at least one unprocessed point p IN sample S DO
2:              Compute N(p) wrt Eps and MinPts.
3:              IF N(p) is null (p is not IN a cluster) THEN
4:                      Label p as noise.
5:              ELSE IF N(p) is density-joinable to an existing cluster THEN
6:                      Merge N(p) and all its density-joinable clusters.
7:              ELSE
8:                      Create a new cluster C based on N(p).
9:              ENDIF
10:     ENDWHILE
```

The complexity of DJ-Cluster and DBSCAN is the same: O(n log n), using a R-Tree based index.

2.2.3 ST-DBSCAN

ST-DBSCAN (BIRANT; KUT, 2007) (*Spatial-Temporal DBSCAN*) is another density-based clustering algorithm based on DBSCAN. While DBSCAN is able to treat only spatial data, ST-DBSCAN is able to deal with non-spatial attributes, like the time. To do it, ST-DBSCAN uses two distance metrics, $Eps1$ and $Eps2$, to define the similarity by a conjunction of two density tests. $Eps1$ is used for spatial values to measure the closeness of two points geographically. $Eps2$ is used to measure the similarity of non-spatial values.

Other two parameters are also used in ST-DBSCAN: $MinPts$ and Δ_E. ***MinPts*** has the same meaning as in the two previous methods, and Δ_E is used to prevent the discovering of combined clusters with little differences in non-spatial values of the neighborhood locations. The existing density-based clustering algorithms are adequate if the clusters are distant from each other, but not satisfactory when the clusters are adjacent to each other. If the values of neighbor objects have little differences, the values of border objects in a cluster may be very different from the values of other border objects in opposite side. In other words, the value of a border object may be very different from the value of the farthest border object, since small value changes on neighbors can cause big value changes between starting points and ending points of a cluster. However, cluster objects should be within a certain distance from the cluster means. This problem is solved by comparing the average value of a cluster with new coming value. If the absolute difference between $Cluster\_Avg$ and $Object\_Value$ is bigger than the threshold value, Δ_E, then the new object is not appended to the cluster. $Cluster\_Avg$ refers to the average or mean value of the objects contained in the cluster. $Object\_Value$ refers to the non-spatial value of an attribute, such as the temperature of a location (BIRANT; KUT, 2007).

The other issue which ST-DBSCAN addresses is related to clusters of different densities. It introduces a new definition of density-distance. It is calculated dividing the density-distance-max by the density-distance-min, which are respectively the largest and shortest distance between p and its neighbors. So, the density factor of a cluster can be calculated. As follows:

$$DensityFactor(C) = 1/\frac{\sum_{p \in C} density-distance(p)}{|C|}$$

The density factor of a cluster *C* captures the degree of the density of the cluster. If *C* is a 'loose' cluster, density-distance-min would increase, and so the density distance would be quite small, thus forcing the density factor of *C* to be close to 1. Otherwise, if *C*

is a 'tight' cluster, *density-distance-min* would decrease, and the density distance would be large, thus forcing the density factor of *C* to be close to 0.

2.2.4 GDBSCAN

GDBSCAN (SANDER et al., 1998) is a generic version of DBSCAN, able to cluster spatial and/or non-spatial objects. DBSCAN is a specialization of GDBSCAN, since only the spatial attributes are used and it 'instantiates' some abstract functions of GDBSCAN. The idea of 'density-based clusters' can be generalized in two important ways. First, we can use any notion of neighborhood instead of an Eps-neighborhood if the definition of the neighborhood is based on a binary predicate which is symmetric and reflexive. Second, instead of simply counting the objects in a neighborhood of an object, we can as well use other measures to define the 'cardinality' of that neighborhood.

Although in many applications the neighborhood predicate will be defined by using only spatial properties of the objects, the formalism is in no way restricted to purely spatial neighbors. We can use non-spatial attributes and combine them with spatial properties of objects to derive a neighborhood predicate.

GDBSCAN uses a kind of weight in the non-spatial attributes to calculate the cardinality of a set of objects. The old Eps-neighborhood function has been replaced by a generic neighborhood function called N_{pred}, which is a symmetrical and reflexive binary predicate. Thus, the *wCard* function calculates a positive value of a set of objects that must be greater or equal than the *MinCard* value in order to be a core object. For instance, the expression wCard(S) $\geq$ MinCard generalizes the condition Eps-Neighborhood(o) $\geq$ MinPts in the definition of density-based clusters, where cardinality is just a special case of a wCard function.

2.2.5 ADBC

Daoying (MA; ZHANG, 2004) proposes an algorithm similar to DBSCAN with some modifications in order to improve the quality of the clustering. The method is called *Adaptive Density-Based Clustering* (ADBC) and the *adaptive* word refers to use an flexible ellipse instead a single circle in order to calculate the neighborhood region. This minimizes the *void pad* and maximizes the *density pad*, which are two concepts related to the cluster quality. They are defined as follows:

Density pad: A density pad is a convex region inside a circle with radius *Eps* that includes all useful objects.

Void pad: A void pad is the region inside the circle with radius *Eps* that is not density pad.

Both definitions are better visualized in the figure 2.15. The larger the void pad is, the higher is the chance to include noise data into a cluster, since for that *Eps*, the cluster assumes a format different from a single circle. Including points in the void region may cause an undesirable chain effect, and the quality of density may be affected. Figure 2.15 shows how the structure of the points may affect the cluster quality according the concepts of void pad and density pad. As the void pad in (b) is larger than in (a), the quality of density in (b) is lower than that (a) (MA; ZHANG, 2004).

The method used to define the neighborhood region is very important to avoid void pad in density-based clustering algorithms. Therefore, the neighboring region has to be defined properly to reflect the data distribution of an object's neighbor. Dynamic ellipse is proposed to adjust with different radiuses and also rotate according to the distribution of the neighbors of an object, as shown in Figure 2.16.

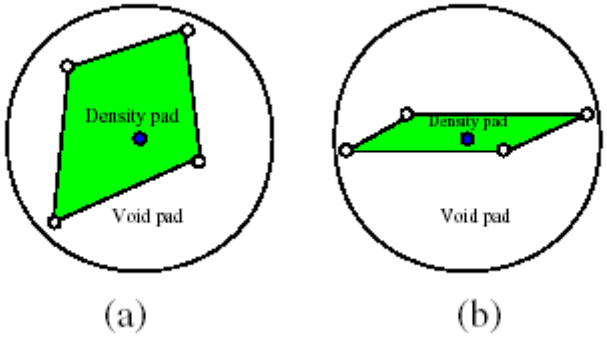

Figure 2.15: *Density Pad* and *Void Pad* concepts (MA; ZHANG, 2004)

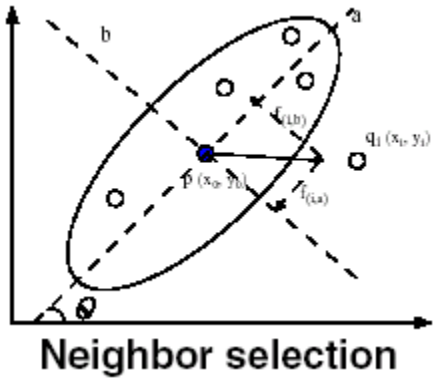

Figure 2.16: The ellipse neighborhood region adapts according to the distribution of the neighbors (MA; ZHANG, 2004)

2.2.6 T-Optics and TF-Optics

Nanni (NANNI; PEDRESCHI, 2006) adapted a density-based clustering method to find trajectory clusters. His work is based on the well known Optics algorithm (ANKERST et al., 1999), following the principle of using the reachability plot of the data in order to identify the clustering structure. Nani adapted Optics to work with trajectories, so he proposed a distance metric to evaluate the similarity among trajectories with respect to a given time interval. The distance metric is the mean of the distances between each pair of points of the same time moment, formally, the continuous mathematical expression is the following:

$$D(\tau_1, \tau_2)|T = \frac{\int_T d(\tau_1(t), \tau_2(t))dt}{|T|}$$

Where τ_i is the trajectory, T is a time window, $\tau_i(t)$ is the point of the trajectory τ_i in the time t and $\|T\|$ is the T's interval duration. Obviously, the computational case will be a discrete set of points, so the integral operator could be replaced by the sum operator $\sum$.

The goal of Nanni (NANNI; PEDRESCHI, 2006) is the discovery of clusters among a set of trajectories with respect to a time window T. The clusters indicate direction and position similarity among the trajectories in the same cluster. In order to achieve it, he

uses a quality function used to measure the clustering result. This function uses the standard high intra-cluster versus low inter-cluster similarity tradeoff principle, which could be naturally translated into high-density clusters versus low-density noise rules. Highly dense clusters can be considered interesting by itself, while having a rarefied noise means that clusters are clearly separated. Put together, these two qualities seem to reasonably qualify a good (density-based) clustering result (NANNI; PEDRESCHI, 2006).

2.2.7 Clustering Considerations

In this section was presented an overview about the clustering algorithms, and some of them were more detailed, like DBSCAN and DJ-Cluster due to their greater influence in this work. The main reason of the choice of DBSCAN in order to be extended to work with spatio-temporal data is due to its simplicity and capacity of find cluster of arbitrary sizes and shapes.

In chapter 3, the proposed model, which is inspired in DB-SCAN, is presented.

3 THE PROPOSED METHOD: CB-SMOT

The main objective of this dissertation is to present a new method to discover important places of trajectories (stops).

SMoT (ALVARES et al., 2007) was the first algorithm proposed in order to discover stops and moves of trajectories, i.e., to represent trajectories in a higher abstraction level. The method proposed in this dissertation, called CB-SMoT (PALMA et al., 2008), extends that work providing a second way to find stops and moves, based on the speed of the moving object.

CB-SMoT stands for *Clustering Based - SMoT* and it uses a clustering technique in order to identify *stops*, according to Spaccapietra's (SPACCAPIETRA et al., 2008) stop definition. CB-SMoT is an algorithm that identifies slower sections in a trajectory and tries to associate them to known places in the background geography, in order to give more human-comprehensive information about the trajectory.

This chapter presents the formal definitions for the method and the pseudo-code for the algorithm. The method has two main steps: initially, using a new clustering method, the important parts of trajectories are identified (section 3.1), and after these parts are compared with the background geographic information to give additional semantics to the clusters (section 3.2).

3.1 Clustering Step

The SMoT (ALVARES et al., 2007) algorithm presented in Chapter 2 matches every trajectory point with every candidate stop in order to extract the stops and moves. However, for some applications, like for instance traffic management, a more important issue is the speed of the moving object.

CB-SMoT intends to create clusters of slow speed parts before matching them with the background geography. So, we use the own trajectories' characteristics (speed) to indicate the most probable interesting parts (sections). The clustering step is inspired on the known clustering method DBSCAN (ESTER et al., 1996), following the same principles but with modifications in the clustering method, since CB-SMoT is based on speed instead of density.

DBSCAN looks for *core points* in order to start a cluster, later it tries to expand it by adding near points. CB-SMoT follows the same principles, first it looks for *core points* and then tries to expand them by aggregating other points in the neighborhood. The main difference is in the core point definition and the aggregating criterion used to expand the cluster. In order to understand these concepts some others have to be introduced:

Definition 6. *Slower-Neighbor*. The Slower-Neighbor of S, $Slower_N(S)$, such that $S =< p_i, p_{i+1}, ..., p_n >$, is the point $p_j = (x, y, t)$ such that:

- $p_j = p_{i-1}$ if $Speed(p_{i-1}) <= Speed(p_{n+1})$ or
- $p_j = p_{n+1}$ if $Speed(p_{n+1}) < Speed(p_{i-1})$

Where $Speed(p)$ is the speed of the point p and S is a time ordered sub-trajectory. If p_{n+1} or p_{i-1} does not exist then the slower-neighbor becomes automatically the remaining point or is not defined if both do not exist.

The other definition is about the neighborhood of a point, called Slowest-Neighborhood of a point p_i. It is calculated by adding to each step, the slowest neighbor until it is impossible adding more points due to the limits of the trajectory, or, when the neighborhood's duration reach the minimum time t. Formally, the slowest neighborhood is defined as follows.

Definition 7. *Slowest-Neighborhood*. The Slowest-Neighborhood of a point p_i with respect to the minimum time t, $SN_t(p_i)$, is defined as:

- $SN_t(p_i) =< p_1, p_2, ..., p_n >$, where $1 \leq i \leq n$ and
- $\frac{\sum_{i=1}^{n-1} d(p_i, p_{i+1})}{t_n - t_1}$ is minimal and
- $t_n - t_1 \geq t$ and $|t_n - t_1|$ is minimal

Where $d(a, b)$ is the euclidean distance between the point a and the point b and $\frac{\sum_{i=1}^{n-1} d(p_i, p_{i+1})}{t_n - t_1}$ is the average speed of the respective subtrajectory.

The Slowest-Neighborhood definition returns the neighborhood that has the minimum speed for a given point p_i. Figure 3.1 presents two examples of slowest-neighborhood. In (a) the starting point is in a high speed section, and all slower-neighbors are to the right, so the final slower-neighborhood will have only the lower speed points, which are at the right side. In (b) the starting point is balanced among other points, so the neighborhood will expand to both sides. Besides, we are interested in a special case of the slowest-neighborhoods: when its average speed is less or equal to a given threshold, avg. The neighborhood will be called *core-neighborhood* if this criterion is matched. Formally, a core-neighborhood is defined as follows:

Definition 8. *Core-Neighborhood*. A Slowest-Neighborhood, $SN_t(p_i)$, is called Core-Neighborhood if $meanSpeed(SN_t(p_i)) \leq avg$, where meanSpeed() is a function that returns the average speed of that neighborhood.

To limit a cluster only by the *MinTime* parameter is not an interesting approach, since a cluster should be a maximal sub-trajectory that satisfies the speed and time constraints. Hence, it is necessary to expand the original core neighborhood by adding other interesting points, which are those points which have a slow speed and are time-close to slowest-neighborhoods. In order to add such points we check the impact of aggregating them in existing core-neighborhoods and comparing the new average speed with the old one. Initially we used only this criterion, but for specific scenarios where the core-neighborhood has a very low average speed, it allows the addition of points whose speed is higher than

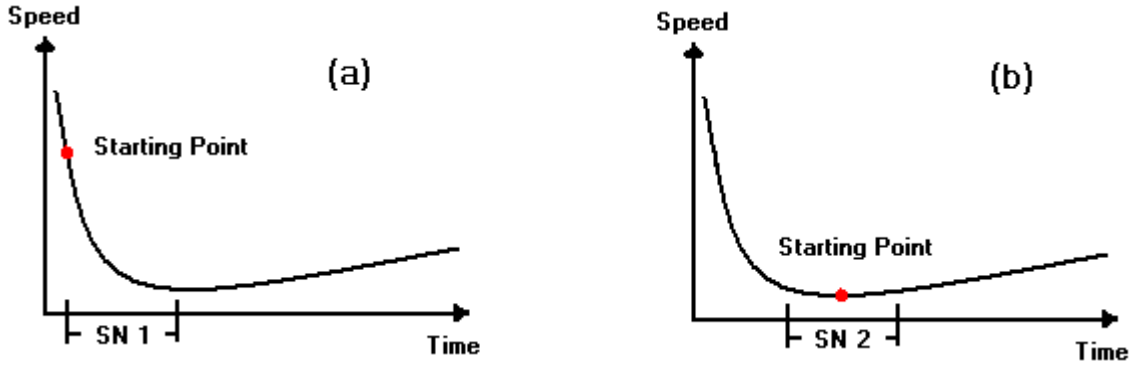

Figure 3.1: Slowest-Neighborhoods created given the starting points

the desirable, but this addition keeps the average speed below the average speed limit, affecting the quality of the final clusters. To avoid the creation of these undesirable clusters, we propose an alternative criterion used to allow a point to be added in a neighborhood. It consists of testing its speed against a upper limit related to the average trajectory's speed. If the point's speed is lower than the limit, the point can be aggregated into the neighborhood. Formally, we use two concepts in order to define this feature: the first one specifies the general way of expanding a neighborhood S. The second uses that definition and limits when it is used.

Definition 9. *Limited-Neighborhood*. The Limited-Neighborhood of S, $LN^{sl}_{avg}(S)$, with respect to the average speed limit avg and the speed limit sl, is defined recursively such that:

- $LN^{sl}_{avg}(S) = LN^{sl}_{avg}(S')$ if $MeanSpeed(S') \leq avg$ and $Speed(p_n) \leq sl$, where $S' = S \cup \{p_n | p_n = Slower_N(S)\}$
- $LN^{sl}_{avg}(S) = S$, otherwise

Where S is a continuous sequence of trajectory points. This definition is not used directly and it is used as an auxiliary definition in order to simplify the next concept, *Connected-Neighborhood*. Definition 9 is about sets of points that have the minimum time duration and the speed criteria satisfied.

Definition 10. *Connected-Neighborhood*. A Connected-Neighborhood of S, $CN^{sl}_{avg,t}(S)$, with respect to minimum time t, the average speed limit avg and the speed limit sl, is defined as:

- $CN^{sl}_{avg,t}(S) = LN^{sl}_{avg}(SN_t(S))$ if $SN_t(S)$ is a core-neighborhood, otherwise is
- $\{\}$ (empty set).

As a core-neighborhood has by definition its average speed less or equal to the avg parameter, the $Limited\_Neighborhood$ used will expand that neighborhood until the speed criterion in definition 8 becomes unmatched. Intuitively, definition 9 connects to a core neighborhood every slower neighbor which keeps the mean speed of the set below

the average speed limit parameter, and tests if the added point is not a non-desirable speed point (when it has a high speed). Figure 3.2 exemplifies how a connected-neighborhood is created. First, we start the set S with a single point shown in (a); then we calculate its slowest-neighborhood in (b), as its mean speed is less than the speed limit parameter it is a core-neighborhood; (c) we expand the core-neighborhood by adding neighbor points that match the criteria used in definition 8.

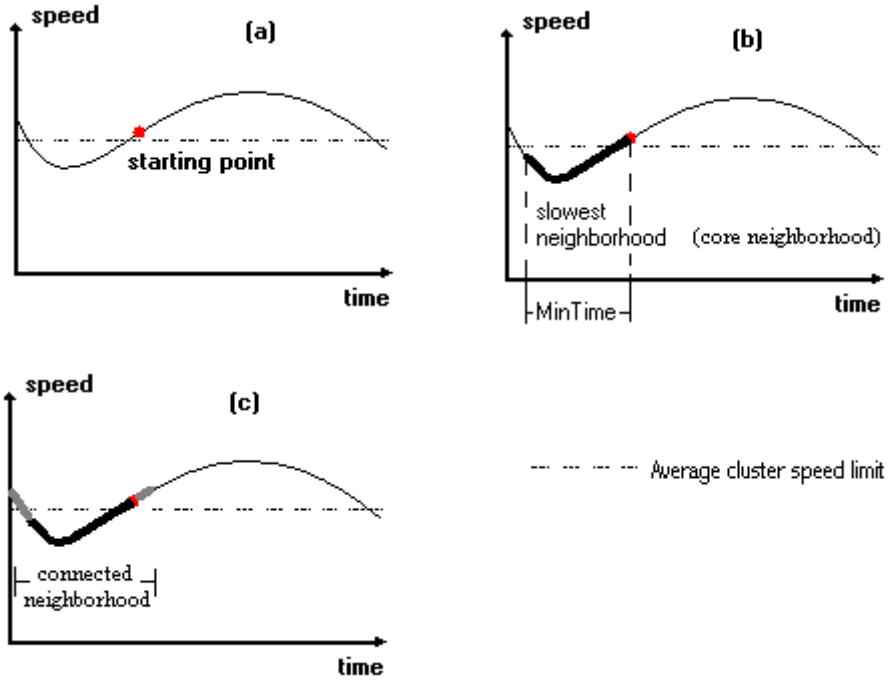

Figure 3.2: Steps of the creation of a *Connected-Neighborhood* given the starting point, minimum time, average speed limit and speed limit

There might exist many connected-neighborhoods in a same timestamp, so we need to choose the best one, in other words, that whose mean speed is the slowest. The last definition chooses among every connected neighborhood the best one in order to be labeled as cluster. The criterion used to compare connected-neighborhoods is the mean speed. So, if two or more connected-neighborhoods share a common point (they intersect each other by this common point), we will call cluster the one whose mean speed is the slowest among them, formally:

Definition 11. *Cluster*. A connected neighborhood CN is called *cluster* with respect to average speed, speed limit and minimum time, if $\forall p \in CN, \forall p' \in CN', CN \neq CN'$ and $p = p' \Rightarrow MeanSpeed(CN) \leq MeanSpeed(CN')$, where CN and CN' are connected neighborhoods of a trajectory.

The first part of CB-SMoT is the clustering step. In this phase we intent to find out slow parts (clusters) in a trajectory in order to use them as input to the next step. Listing 3.1 shows the main steps of CB-SMoT in a pseudo-code.

Listing 3.1: CB-SMoT Clustering Algorithm

```
void SpeedClustering(T,avg,MT,sl)

INPUT:
T: Trajectory
avg: average cluster speed limit
MT: cluster minimum time duration
sl: cluster speed limit

METHOD:
1:clusterId = 1;
2:T1 = T.sortBySpeed();
3:FOR EACH p IN T1 DO
4:      IF p is UNPROCESSED THEN
5:              IF Limited-Neighborhood(T,p,clusterId,avg,MT,sl) THEN
6:                      clusterId++;
7:ENDFOR

OUTPUT:
EACH p IN T labeled as the respective cluster id identification
```

The inputs are the trajectory points, the avg, MT and sl parameters. The sl parameter is the speed limit of a candidate point that will be added into an existing neighborhood created from the other two parameters. The *clusterId* is a cluster identifier that is incremented each time a cluster is found. In line 2 the points of the trajectory are sorted by ascending speed in order to be iterated at line 3. Therefore, we try to generate clusters with the slowest points before. If a point has already been processed (it was set to a cluster by the Limited-Neighborhood function) we jump to the next one.

The *Limited-Neighborhood* function (in line 5 of listing 3.1) is detailed in listing 3.2, and works as follows. First (line 4), it tests if the slowest-neighborhood of a point *p* is a core-neighborhood. If it is not, the function returns false and the next point in the main loop is tested; otherwise its neighborhood is set with the given $clusterId$, and we try to expand the cluster in the loop of line 8. Hence, we add one point each time into the new generated cluster, testing the criteria in definition 8. This test warrants the quality of the cluster, ensuring that always the cluster will have its mean speed less or equal to the avg parameter and it will not have any point whose speed is too high.

Listing 3.2: Limited Neighborhood Function

```
boolean Limited-Neighborhood(T, p, clusterId, avg, MT, sl)

1:Seeds = {p};
2:Slowest-Neighborhood(Seeds,T,MT);
3:MeanSpeed = Seeds.meanSpeed();
4:IF (MeanSpeed > avg OR Seeds.duration() < MT)
5:      return false;
6:ELSE
7:      Seeds.setClusterId(clusterId);
8:      WHILE (true)
9:              AddedPoint = Slower-Neighbor(Seeds,T);
                //Add the point at time ordered correct position
10:             Seeds.add(AddedPoint);
11:             NewMeanSpeed = Seeds.meanSpeed();
12:             IF (NewMeanSpeed <= avg AND AddedPoint.speed() <= sl)
13:                     IF (AddedPoint is not IN a cluster) {
14:                             AddedPoint.clusterId = clusterId;
15:                     ELSE
16:                             break;
17:             ELSE
18:                     break;
19:             MeanSpeed = NewMeanSpeed;
20:     ENDWHILE
21:ENDIF
```

```
22:return true;
```

The Slowest-Neighborhood function shown in listing 3.3, basically compares the speed of the points immediately before and after of a given time-ordered set of points (the initial calling set is the singleton with the point p) and adds the point whose speed is slower. This process is repeated until the MT duration is reached or is impossible to add more points into the set due to the limits of the trajectory or the points are already in another cluster.

Listing 3.3: CB-SMoT Slowest-Neighborhood Function

```
void Slowest-Neighborhood(Seeds, T, MT)

1:Do
2:      leftPoint = T[Seeds.firstPointIndex()-1];
3:      rightPoint = T[Seeds.lastPointIndex()+1];
4:      IF (leftPoint.speed <= rightPoint.speed AND
                        leftPoint is not IN a cluster)
5:              Seeds.add(leftPoint);   //Adds in the correct position
6:      ELSE IF (rightPoint is not IN a cluster)
7:              Seeds.add(rightPoint); //Adds in the correct position
8:      ELSE
9:              break;
10:WHILE (Seeds.duration() < MT);
```

Each point in the trajectory is analyzed only once, in order to label it as a cluster or not, so the complexity of the clustering step is N (the number of points in the trajectory). In order to find the neighbors of a given point it is only necessary to increment/decrement the index in the trajectory, since the trajectory contains all points ordered in time.

3.2 Giving Semantics to the Clusters

The slower parts of the trajectory are discovered in the clustering step and labeled as clusters. Those parts are interesting trajectory's parts, so they are labeled as *stops* according to our approach. However, we will divide them in two kinds: *Unknown Stops* and *Known Stops*. In order to formally define these concepts we will use two other concepts proposed by Alvares (ALVARES et al., 2007): *Candidate Stop* and *Application*, which were defined in the previous chapter.

Definition 12. *Known Stop*. Let T be a trajectory with time domain I and let $A = (\{C_1 = (R_{C_1}, \Delta_{C_1}), ..., C_N = (R_{C_N}, \Delta\_C_N)\})$ be an application. Let $I_0, I_1, ..., I_n$ be the temporally-ordered decomposition of I into time intervals where T is intersected by candidate stops with respect to A. A known stop of T with respect to A is a contiguous sub-trajectory of T over a maximal concatenation of some $t_i, t_{i+1}, ..., t_{i+\ell}$ for which there is a (R_{C_k}, Δ_{C_k}) in A such that:

- $< (x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1}), ..., (x_{i+\ell}, y_{i+\ell}, t_{i+\ell}) >$ is in a cluster
- $< (x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1}), ..., (x_{i+\ell}, y_{i+\ell}, t_{i+\ell}) >$ intersects R_{C_k}.
- $|t_{i+\ell} - t_i| \geq \Delta_{C_k}$

An *Unknown Stop* is related to clusters (or cluster parts) that do not have intersections that lasts MT with the same candidate stop with respect to the application. Thus, the formal definition is the following:

Definition 13. *Unknown Stop*. Let T be a trajectory with time domain I and let $A = (\{C_1 = (R_{C_1}, \Delta_{C_1}), ..., C_N = (R_{C_N}, \Delta\_C_N)\})$ be an application. An unknown stop of T with respect to A is a contiguous sub-trajectory of T over a maximal concatenation of some $t_i, t_{i+1}, ..., t_{i+\ell}$ such that:

- $< (x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1}), ..., (x_{i+\ell}, y_{i+\ell}, t_{i+\ell}) >$ is in a cluster
- There is no point $p_n, i <= n <= i + \ell$, such that p_n is in a Known Stop
- $|t_{i+\ell} - t_i| \geq MT$

There are some characteristics related with Known and Unknown stops: both definitions require that the points belong to a cluster, but a *Known Stop* may have points in more than one cluster. Such extreme case occurs when the next point after the last one of the first cluster is the first point of the next cluster, shown in Figure 3.3 (a)[1]. In other words it means that all consecutive points that intersect a candidate stop are in a cluster. Another peculiar case is related to *Unknown Stops* which do not necessarily use all points in the cluster, because there are points in the cluster that belong to *Known Stops*, shown in Figure 3.3 (b).

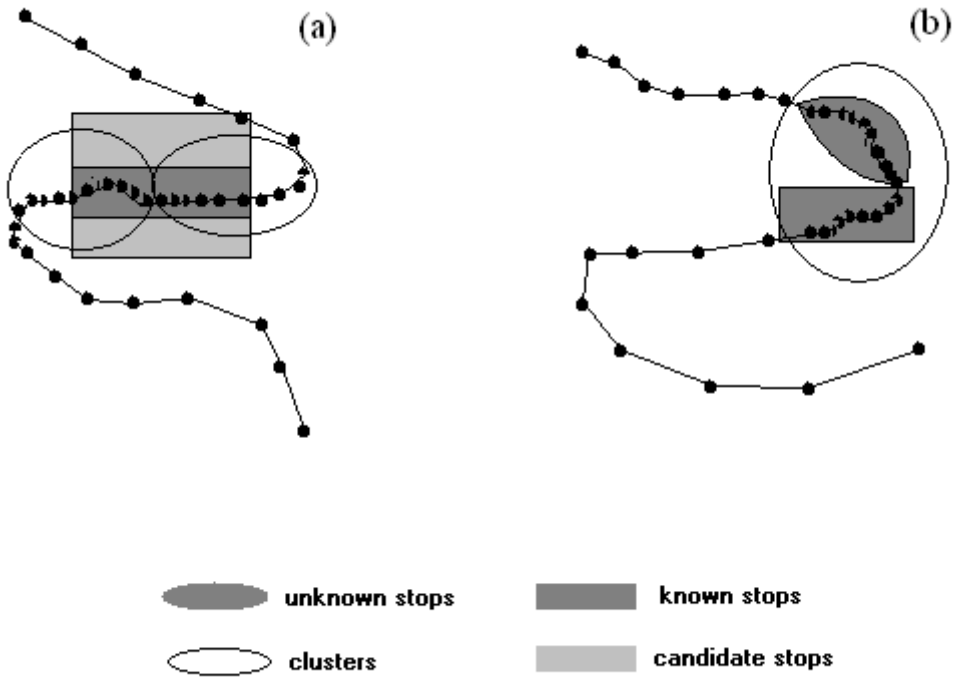

Figure 3.3: Some possible cases of Known Stops and Unknown Stops. In (a) a Known Stops may be in more than one cluster. In (b) a cluster that generates a known stop and a unknown stop

[1]In order to facilitate the visualization of the clusters, known stops and unknown stops were represented as shapes, but in fact they are only the points inside these shapes. Only the candidate stop has an area associated to it

The points which will form stops must necessarily be in a cluster. After found the clusters, we identify *Known Stops*. For last we try do create *Unknown Stops* with the remaining points. The remaining points after the discovery of the *Known Stops* will not necessarily generate an *Unknown Stop*, since some criteria must be matched, like the minimum time duration. A necessary condition to generate stops is that all points in the same stop must be consecutive. In other words, there may not be any trajectory point between two points in the same stop that is not in that stop.

3.2.1 Stops Discovering Algorithm

The second step of CB-SMoT is to try to associate the clusters found in the clustering step with some features in the background geography. Some clusters or parts of clusters will intersect candidate stops, and they will be called *Known Stops*. Clusters without any mapped relation with the considered background geography will be labeled as *Unknown Stops*.

The clustering step acts like a filter, selecting those trajectory points that we are interested in, i.e., parts with low speed. The next step is to test if the cluster will intersect a candidate stop for the respective minimum stop duration, otherwise the cluster will be labeled as *Unknown Stop*.

This part is quite simple, but we need some care about the cases presented in Figure 3.3. A simpler way to do that is to treat adjacent clusters (like in Figure 3.3(a)), as a single bigger cluster, despite they have been created at different moments. Therefore, each time we create a new stop, we associate to it the cluster identifier of the point which intersected it. So, each new point that would be in the same known stop must have the same cluster identifier. In other words, they must be in the same cluster. This warranty that all points in the same stop will be consecutive.

Another possible case is when there are two clusters with at least one point (which is not in a cluster) between them, and a candidate stop whose intersection starts in the first cluster, continues through this point, and it finishes in the next cluster. Such candidate stop will not generate a *Known Stop* because there is no continuous intersection with cluster points. This case is exemplified in Figure 3.4[2]. In this example, we are considering that the minimum time duration associated to the candidate stop is not satisfied in any cluster separately. In other words, the sum of both cluster intersection time is not used in order to test the intersection time with that candidate stop in order to generate a *known stop*. However, if the minimum time duration is satisfied in each cluster separately, we will have two known stops.

In listing 3.4 we present the complete high-level algorithm with the clusterization step and the stops discovery.

Listing 3.4: CB-SMoT

```
void CBSMoT(T,avg,MT,sl,A)

INPUT:
T       :  Trajectory
avg:  Average Speed
MT :  MinTime
sl :  SpeedLimit
A  :  Application

1:SpeedClustering(T,avg,MT,sl);
```

[2]the cluster was presented as a circle in order to facilitate the visualization, but it is formed by only the points inside the circle

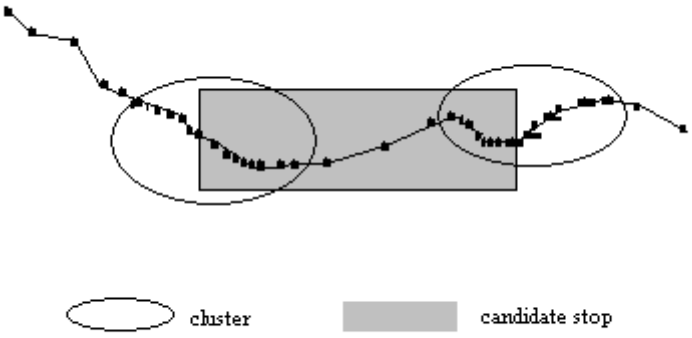

Figure 3.4: The candidate stop will not form a *Known Stop*, since it has not a continuous intersection with cluster points and the minimum time duration is not satisfied in none of them

```
//Consider adjacent clusters as a bigger single one
2:T.unifyAdjacentClusters();
3:Points = T.clusterPoints();
4:FOR EACH p IN Points DO
5:      IF p intersects some candidate stop C THEN
6:              List.add(new Association(p,C));
7:      ELSE
8:              List.add(new Association(p,null));
9:              ENDIF
10:     ENDFOR
//Find the Stops
12:StopsDiscovering(List);
```

At line 1 the method performs the clustering step, but it changes the cluster identification at line 2 in order to treat the adjacent clusters as a bigger single cluster. After (lines 4-10), it iterates through the points in a time ascendant order creating a list that associates each point with the candidate stop it intersects if it intersects any. By definition, an application has no candidate stops that overlap each other, so a trajectory point intersects either none or one candidate stop only. The algorithm is presented this way in order to facilitate the understanding, but it may be easily extended in order to allow candidate stops overlapping. The *Association* structure binds each point with the candidate stop(s), if they exist.

Listing 3.5: StopsDiscovering

```
void StopsDiscovering(List)

INPUT:
List     :  List of points associated with candidate stops

1:KnownStop = null; UnknownStop = null;
2:FOR (i=0;i<List.size;i++) DO
3:      IF List[i].C != null THEN
4:              IF List[i].C != KnownStop.C or
                KnownStop.clusterId != List[i].p.clusterId THEN
5:                      IF KnownStop.duration >= MT THEN
6:                              KnownStops.add(KnownStop);
7:                      ENDIF
8:                      KnownStop =
                        new KnownStop(List[i].C,List[i].p.clusterId);
9:              ENDIF
10:             KnownStop.add(List[i].p);
```

```
11:     ELSE
12:             IF UnknownStop.clusterId != List[i].p.clusterId THEN
13:                     IF UnknownStop.duration >= MT THEN
14:                             Unknowns.add(UnknownStop);
15:                     ENDIF
16:                     UnknownStop =
                        new UnknownStop(List[i].p.clusterId);
17:             ENDIF
18:             UnknownStop.add(List[i].p);
19:     ENDIF
20:     IF KnownStop.duration >= MT THEN
21:             KnownStops.add(KnownStop);
22:     ENDIF
23:     IF UnknownStop.duration >= MT THEN
24:             Unknowns.add(UnknownStop);
25:     ENDIF
```

The StopsDiscovering procedure, shown in listing 3.5, is used to generate *Known and Unknown Stops*. The algorithm shows the case used when it is allowed at most one candidate stop by point. Whether more than one candidate stop is allowed, the procedure needs to treat them separately, but this case will not be presented here. At the beginning of the StopsDiscovering procedure two empty lists are created: *KnownStops* and *Unknowns*, which will be filled along the procedure. The *IF* in line 3 tests if the point has an associated candidate stop. In positive case it may become a *known stop*, so the next *IF* (line 4) tests the conditions required in order to it not be added in the existing KnownStop (if it exists). If these conditions are not satisfied (it implies it belongs to the same *KnownStop*), then the point is added to the existing *known stop* in line 10. Otherwise, it is necessary to create a new *KnownStop*, but before that, we need test if the previous *KnownStop* is in fact a *KnownStop* by matching its minimum time duration and adding it to the *KnownStops* list in positive case (line 6). Adding it or not, it is necessary to create another *KnownStop*, and it is done in line 8, where we pass to the constructor the candidate stop and the cluster identification.

In case the test in line 3 is false, in other words the point has no candidate stops associated to it, it only may become an *UnknownStop*. Thus, the next *IF* (line 12) tests if the point's cluster identifier is the same of the existing *UnknownStop* (if it already exists). In positive case, the point is added to it (line 18), otherwise it is needed to test if the previous one matches the minimum time duration (line 13) in order to add it to the *UnknownStops* list. Matching this criterion or not, another *UnknownStop* needs to be created (line 16), where only the cluster identification is passed into the constructor.

In lines 20 to 25 we test the pending *KnownStop* and *UnknownStop* in order to add them into the respective lists, because they might not have being added due to the end of the looping.

Considering processing time, the worst case in this part of the algorithm occurs when every point is in a cluster. Thus, every point will need to be tested with the background geography and the complexity becomes the same of the SMoT algorithm: *O(N Log C)*, where *N* is the number of points in the trajectory and *C* is the number of candidate stops. The final complexity of CB-SMoT in the worst case is given by the sum of the complexity of the clustering step added to the complexity of the matching with the background geography. So it is *O(N + N Log C)*, which is *O(N Log C)*.

3.2.2 Common Unknown Stops Among Trajectories

The algorithm CB-SMoT, presented in this chapter, is used to find *known* and *unknown stops* in a single trajectory. However, a more interesting approach is to use CB-SMoT in a trajectory dataset and try to identify common patterns of stops. When we discover *known*

stops, each one is related to a place in the background geography, so that, if two *known stops* of different trajectories take place approximately in the same location, they might be related to the same geographic entity [3]. In a similar way, the *unknown stops* may also refer to the same geographic place.

We proposed a similar process in order to try to relate unknowns to each other, since often *unknown stops* that take place approximately in the same location refer to the same geographic entity or event. This approach is interesting later, when the CB-SMoT output will be used to discover new patterns. Therefore, instead of trying to discover patterns by treating each unknown in an isolate way, which will generate no patterns, we associate them to each other by the same identifier.

The identification process is performed after applying CB-SMoT in the whole trajectory dataset. It assigns identification *id* to the unknown stops. The same *id* is set to all unknowns that are close to each other. The closeness criterion is based on the topological intersection operation, i.e., for each unknown stop *U*, if it intersects [4] another unknown stop which has already an id, *U's* id becomes the same, otherwise *U* receives a new identification. Figure 3.5 illustrates this case, where *unknown 2* is a stop in both trajectory *A* and *B*.

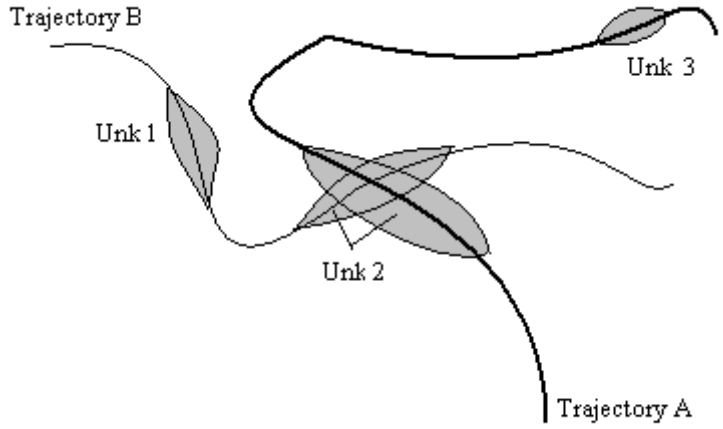

Figure 3.5: The *unknown stops* that intersect each other will have the same identification

[3]The term *entity* is used to define some physical place in the background geography

[4]We define the shape of an unknown as a buffer involving the sub-trajectory represented by a cluster.

4 A PROTOTYPE: WEKA-STPM

In order to evaluate the proposed model, we implemented it as an extension of Weka (WITTEN; FRANK, 2005), which is a well known open source tool that implements many data mining algorithms. Weka is used in several groups in order to perform traditional data mining. Our research group had already extended Weka to pre-process geographical databases (BOGORNY et al., 2006). Now, Weka was extended in order to pre-process spatio-temporal data by the addition of the Semantic Trajectory Pre-process Module (STPM), which is described in this chapter.

4.1 STPM Interface

In order to support STPM we extended the Weka database connection interface. We added the button *'Trajectories...'*, shown in figure 4.1, which calls the STPM module.

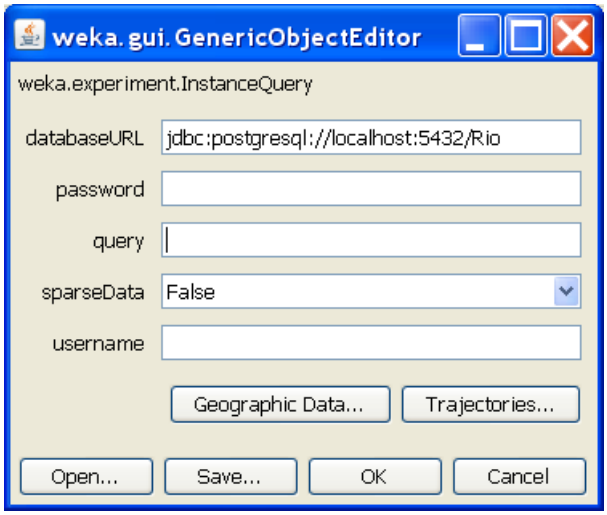

Figure 4.1: Screen in Weka where the STPM is called

Figure 4.2 shows the visual interface of STPM. At the top of the interface, the user needs to choose the schema in the database. After click on 'load', all geographic tables in that schema will be loaded into *Trajectory Table* combobox and into the *RelevantFeatures* list. In the combo 'Trajectory Table' the user should choose the trajectory table that will be considered. In the list 'Relevant Features', the user should choose the candidate stops. In the granularity panel the user will define in which granularity level the output data will be generated. *Feature Instance* level indicates that he is interested in the 'instances'/objects of the relevant features. *Feature type* level indicates that he is interested in the *class/type*

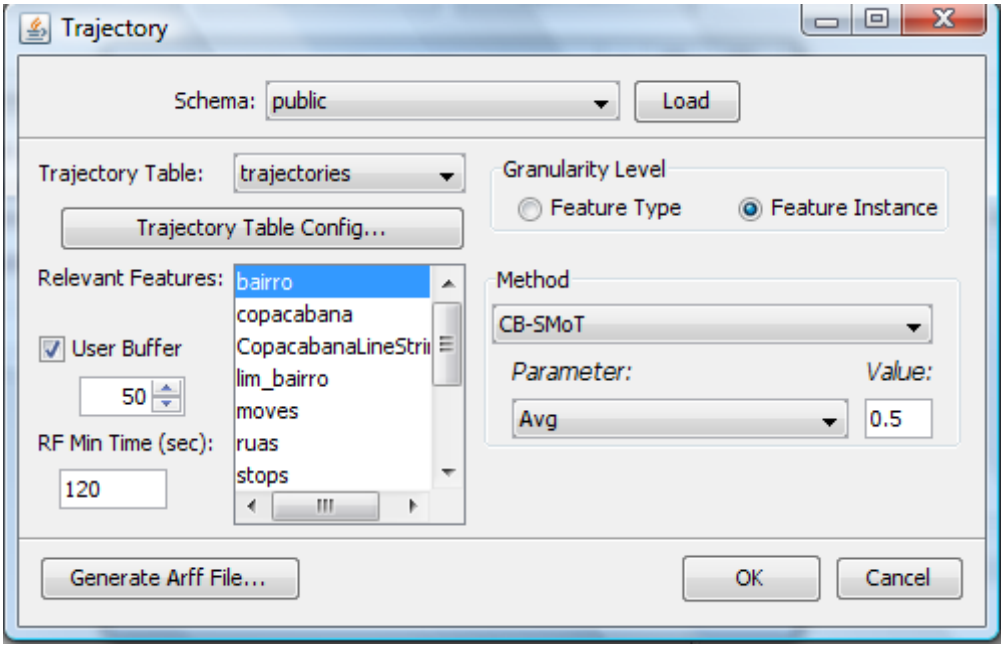

Figure 4.2: STPM User Interface

of the relevant feature. For instance, if the user chooses as relevant feature 'hotel', the instance level will generate an output like 'hotel_Abadia', 'hotel_Sheraton', etc. On the other hand, the feature type granularity will generate only the class as output, in that case 'hotel'. For more details, please see (BOGORNY et al., 2007). The parameter *User Buffer* defines the size of the buffer operation that will be used for relevant features represented as a point or line, in order to transform them in a polygon and them avoid problems of imprecision. The *RF Min Time* parameter defines the minimum time duration in order to a relevant feature be considered a stop. In the *method* panel, the user may choose which method to use to generate semantic trajectories: SMoT or CB-SMoT, and define its respective parameters.

The *Trajectory Table Config* button was designed to adjust the trajectory table considering some limitations of the prototype. This table should have an attribute called *Tid* of type integer that is used as the trajectory identification, and the attribute that stores the time information of the trajectory point should be called *time* and be of type timestamp. Figure 4.3 shows the interface to configure the trajectory table. It presents two examples of scripts that can be adapted to these transformations and can be executed from this interface. One of the scripts may be used in order to generate the numeric *tid*, based on any trajectory identification type. The other one may be used in order to generate a time of type timestamp, based on a numeric field that represents the time dimension.

The *Generate Numeric Tid* script is presented in listing 4.1. First it creates a new sequence and a temporary table. The next step is to select all distinct values of the trajectory identificator field selected by the user and put them into that table associating it with a number of that sequence. After, a 'tid' column is added in the trajectory original table and an update is performed by looking the respective identification of the record and updating the new 'tid' field with the sequence number existing in that. Last, the sequence and the temporary table are dropped.

Listing 4.1: Generate Numeric Tid

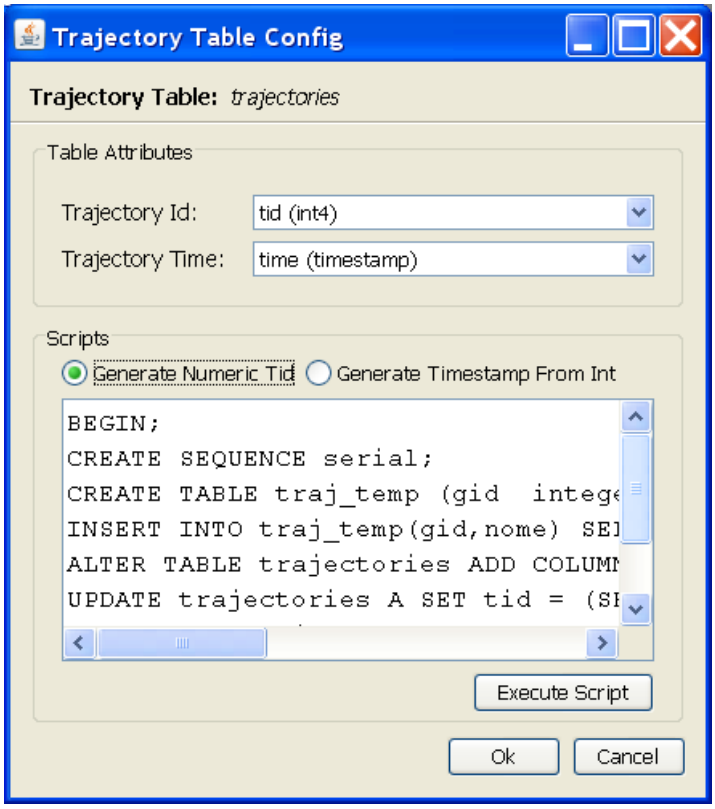

Figure 4.3: Trajectory table configuration screen

```
BEGIN;
CREATE SEQUENCE serial;
CREATE TABLE traj_temp (gid  integer,nome character varying) WITHOUT OIDS;
INSERT INTO traj_temp(gid,nome)
        SELECT DISTINCT ON (tipo) nextval('serial'), tipo FROM trajectories;
ALTER TABLE trajectories ADD COLUMN "tid" integer;
UPDATE trajectories A SET
        tid = (SELECT t.gid FROM traj_temp T WHERE T.nome = A.tipo LIMIT 1);
DROP TABLE traj_temp;
DROP SEQUENCE serial;
COMMIT;
```

The *Generate Timestamp from Int* script is simpler, since it is a more specific script, as shown in listing 4.2. It must be used when the time format is an integer. First it adds the time attribute in the trajectory table. After it uses a composed function which 'casts' the integer in a timestamp format. Of course, these scripts should be adapted according to the input data.

Listing 4.2: Generate Timestamp from Int

```
BEGIN;
ALTER TABLE trajectories ADD COLUMN "time" timestamp without time zone;
UPDATE trajectories SET time = CAST(('2006-06-'||
        (1 + (substr(time,1,6)::int)/86400) ||' '||
        (substr(timeStr,1,6)::int)\%86400/3600 ||':'||
        (substr(timeStr,1,6)::int)\%86400\%3600 / 60 || ':' ||
        (substr(timeStr,1,6)::int)\%86400\%3600 \% 60) as timestamp);
COMMIT;
```

The output of STPM module are two tables in the database: a stop table and a move table. The stop table contains all stops (*Knowns* and *Unknowns*) found. Each record in this table has the stop identification, the stop's *start_time*, *end_time* and its geometry.

The stop identification is composed by the trajectory identification (tid), the stop number of that trajectory (stopid) and it has associated a stop name(*stop_name*), which can be feature type or feature instance granularity level. The move table contains in each record the detailed description about a move, the trajectory identifier, the start and end time, the start stop and the end stop (if they exist).

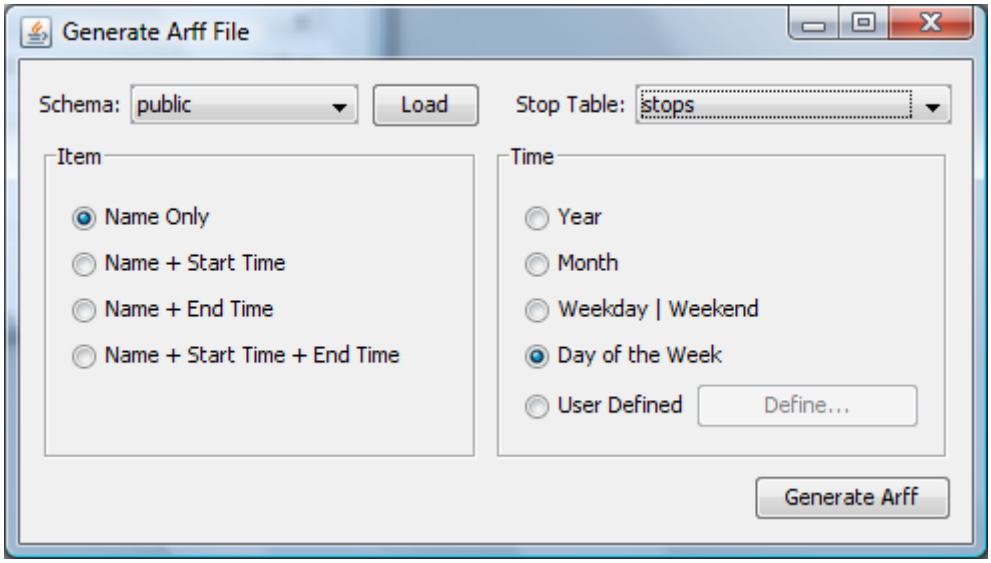

Figure 4.4: Generate *arff* file interface

Besides, STPM can create *arff* files (native Weka text files) to be used by Weka for data mining purposes. This is done by the *Generating Arff File* button in figure 4.2. Figure 4.4 shows the interface with the parameters necessary to generate the arff file. In that screen the user chooses the stop table of the respective database schema and defines how the item will be generated by combining the elements in the *Item* panel and the *Time* panel, i.e., defining the type of aggregation used. *Name Only* specifies that time will not be considered. When time is considered, the *Time* panel specifies how the time will be aggregated. This interface allows the user to aggregate the data in different granularity levels, what is very important for data mining. For details, please see (BOGORNY; KUIJPERS; ALVARES, 2008).

Arff files have two distinct sections. The first section is the Header information, which is followed by the Data information. The Header of the *arff* file contains the name of the relation and a list of the attributes (the columns in the data), and their types. The data body contains the information of each instance of data, where each column is an attribute of the header. An example of an *arff* file looks like the listing 4.3.

Listing 4.3: Arff File

```
@RELATION Stops

@ATTRIBUTE tid NUMERIC
@ATTRIBUTE 139_bairro {yes}
@ATTRIBUTE 140_bairro {yes}
@ATTRIBUTE 64_bairro {yes}
@ATTRIBUTE 1_unknown {yes}
@ATTRIBUTE 152_bairro {yes}
@ATTRIBUTE 151_bairro {yes}
@ATTRIBUTE 149_bairro {yes}
@ATTRIBUTE 150_bairro {yes}
@ATTRIBUTE 109_bairro {yes}
```

```
@ATTRIBUTE 154_bairro {yes}
@ATTRIBUTE 41_bairro {yes}
@ATTRIBUTE 39_bairro {yes}
@ATTRIBUTE 0_unknown {yes}
@ATTRIBUTE 60_bairro {yes}
@ATTRIBUTE 19_bairro {yes}
@ATTRIBUTE 65_bairro {yes}

@DATA
873,?,?,?,?,?,?,?,?,?,?,?,yes,?,?,?,?
804,?,yes,?,?,?,yes,yes,yes,?,yes,yes,?,yes,?,?,yes
802,?,yes,?,?,?,yes,yes,yes,?,yes,yes,?,?,?,yes,yes
800,?,yes,?,?,?,yes,yes,yes,?,yes,yes,?,?,?,yes,yes
846,yes,yes,?,yes,yes,yes,yes,yes,yes,yes,yes,?,?,?,yes,yes
799,?,yes,?,?,?,yes,yes,yes,?,yes,yes,?,?,yes,?,yes
798,?,?,yes,?,?,?,?,?,?,?,?,yes,?,?,?,?
844,yes,yes,?,yes,?,yes,yes,yes,?,yes,yes,yes,?,?,?,yes
843,yes,yes,?,yes,yes,yes,yes,yes,?,?,yes,?,?,?,yes,yes
```

4.2 STPM Implementation

Figure 4.5 shows the main components of STPM. Most of the STPM functionality is provided by the Functional Module. It uses the data read by the interface, and performs the processing using the data in the appropriated format of STPM. This module contains static methods which are called from the interface. The main methods of this module are:

- ***smot***: performs the SMoT algorithm given the trajectory, the application and the parameters.
- ***speedClustering***: is responsible for clustering the given trajectory according the given parameters. It corresponds to the first step of CB-SMoT.
- ***stopsDiscovery***: gives semantics to the clusters. Corresponds to the second step of CB-SMoT.
- ***saveStops***: It is a method used to save into the database the stops found by CB-SMoT or SMoT method.

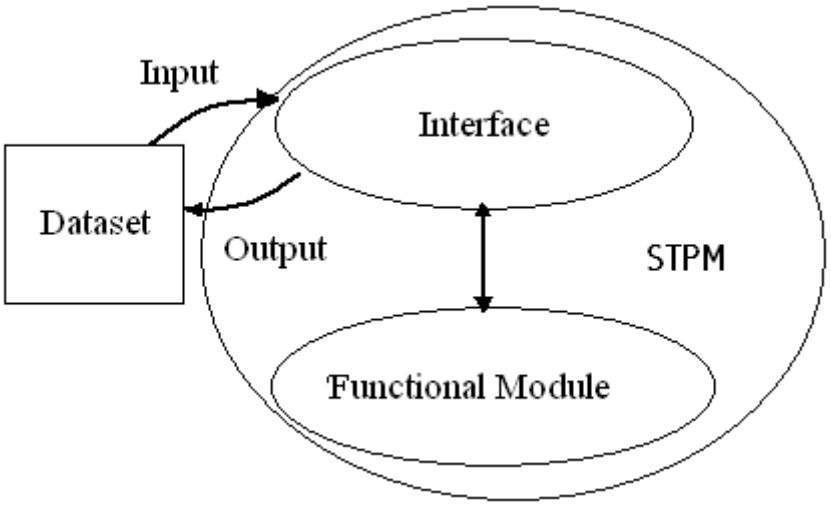

Figure 4.5: Main STPM Components

An important part used in the communication between the interface and the functional module is the structure GPSPoint, shown in Figure 4.6. It represents a class with the information of the points read from the trajectory and it has some methods used in the functional module. It is built in the interface module when the trajectory data is loaded. When the clustering method is performed, the clustering information is aggregated to this structure, which contains the essential data for the communication between the two main modules.

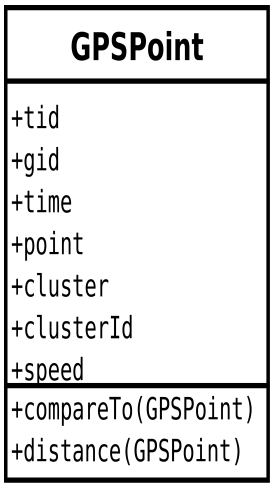

Figure 4.6: GPSPoint high level class

The attributes of the GPSPoint class are:

- *tid*. the trajectory identification;
- *gid*. the geographic identification;
- *time*. the timestamp of the point;
- *point*. the coordinates of the point;
- *cluster*. an enumeration indicating if the point belongs to {CLUSTER, NONE, MOVE};
- *clusterId*. an identification of the cluster if it is in a cluster;
- *speed*. the speed of the point.

The methods are *compareTo(GPSPoint)*, which is used in order to sort the points by *tid, time*; and *distance(GPSPoint)*, which is used in order to measure the spatial distance to the given GPSPoint.

The insertion of STPM into Weka, follows the same principle used to insert GDPM into Weka. The PropertySheetPanel file is inserted in the GUI package of Weka. Figure 4.7 shows how the STPM module is inserted in the Weka program structure. The PropertySheetPanel is a class in the Weka package and this file has been modified in order to aggregate the STPM. It was modified by adding an instanceQueryAdapter class at the end of the file, which is responsible to instanciate the STPM main class, passing the respective parameters. Figure 4.8 shows how this class calls the STPM main class. The instanceQueryAdapter needs the PropertySheetPanel in order to see its scope and be able to pass to the TrajectoryFrame the three parameters used to connect to the database: *user*, *pass* and *url*. After this, the TrajectoryFrame is an independent program.

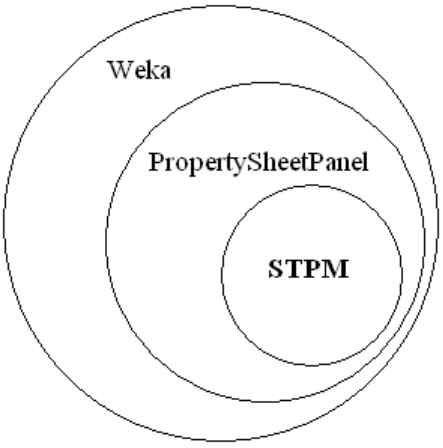

Figure 4.7: Where find the STPM in Weka java source

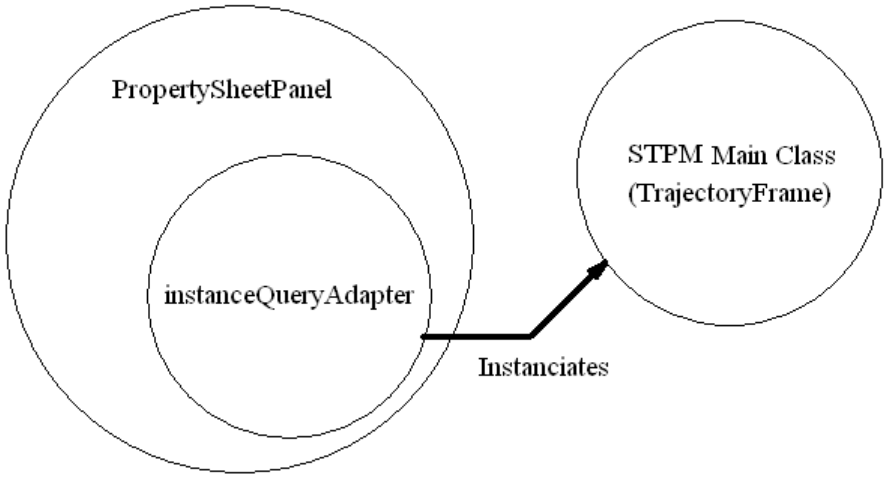

Figure 4.8: How STPM main class is instantiated from the PropertySheetPanel class

4.3 Project Decisions

As used to happen, first a theoretical model is proposed, and in the implementation process some decisions need to be taken in order to fit the real world. In the theoretical model, Alvares (ALVARES et al., 2007) limits some aspects like, for instance, the relevant features can not overlap each other. This approach is useful because it facilitates the precise definition of stops and moves. However, not all relevant features are non-overlapped. For instance, street is a relevant feature that may cross other streets.

To solve this problem, we decided to consider only the first relevant feature that intersects a point of a cluster, i.e, when a point of a cluster intersects a relevant feature, the verification of intersection is finished (the intersection is not tested with other relevant features). With this decision, we consider that a relevant feature instance does not intersect any other, but if it does, the program will ignore this fact.

The *Known Stops* geometric identification has changed along the cycle of development. Initially, the *Known Stops* geometry was the own candidate stop, but it is impracticable when we work with streets being a single line. When a trajectory intersects a

street, if we associate the whole street as geometry, we will loose the information about which part of the trajectory in fact intersected the street. So, we decided to create a buffer around the points which intersected the street in order to be more precise to locate where the intersection occurred. Then, we can visually identify the stops.

To give a label to *Unknown stops* is not a trivial task. When an *Unknown stop U* that will be labeled (receive an identifier) intersects two or more distinct and already labeled *unknows* (e.g. *unknown-1* and *unknown-2*), we need to decide which label of the already labeled *unknowns* we will use for *U*. It was a difficult choice, because many considerations have to be taken into account, such as the greatest intersection, the closest centroids, etc. Hence, we opted for giving the minor integer identification by its simplicity (e.g. *unknown-1*).

The visual interface was projected in order to be as generic as possible. Application definition consists in a set of relevant features, thus we opted by loading all relevant features from the database and let the user choose which ones he will use. In other words, the user will define his own applications given the available relevant features.

As some spatial operations are easily performed inside the database and other ones are easily programed in an outside program (the STPM module), some decisions were taken in order to facilitate the programming and accelerate the program development. The clustering procedure was developed outside of the database, since a more flexible and high level language (we used the Java language) would let the programing task easier than programming in a database language. However, some aspects are inherent from geographic databases, like the geographic operations. Thus, in the geographic operations we took advantage of the database facilities (spatial SQL) and the STPM was programmed using its facilities. Thereupon, the STPM module alternates the processing between the program written in Java and the facilitates provided by the database. This decision accelerated the program development but it let it not completely optimized. The best approach probably would be implement the whole process in one side only.

5 EXPERIMENTS

This chapter presents the experiments performed to validate the proposed method. In these experiments we used trajectory data collected in Rio de Janeiro city, by the Traffic Engineering Company of Rio de Janeiro. This dataset contains more than 2 thousand trajectories. Initially, we had to prepare the trajectory data to fit our needs. This was not an easy task. For instance, the raw data were obtained from the real world by a GPS device. However, the trajectories were not clearly delimited, and we needed to limit them fixing an amount of time as an interval from one trajectory to another. This time interval was defined as 1 hour. After identify the trajectories a problem was pointed. Sometimes, the GPS was turned off, and then turned on, after a period less than 1 hour, in another place. This fact generated a straight line in the trajectory, which might pass through mountains or other places where there are no streets. As relevant features we have some elements of the Rio de Janeiro's dataset. The elements used are a table of streets in Rio de Janeiro and the table of districts. The districts table was according our specifications and we had no problem to treat with them. However the streets table was not according the specifications. Each record in that table was a section in the street and not the whole street and we need to adjust that table in order to prepare it for our experiments.

5.1 Clustering Parameters

The clusters were formally defined in section 3, but it is difficult to formally evaluate the quality of a cluster. Many clustering algorithms are evaluated through a visual representation. In a similar way, we use a visual representation of the trajectory and present the clustering result in a visual representation as well. Obviously clusters are important in order to show the quality of the clustering method. However, these clusters should not be obviously delimited in order to evaluate the influence of the clustering parameters in their quality.

According to the application domain, the values of the parameters *average speed* (avg) and *speed limit* (sl) will change, i.e., the most appropriate values of them for one trajectory dataset may not be the same for another dataset. Trying to minimize this problem, instead of using an absolute value for avg and sl (for instance $avg = 30km/h$ and $sl = 40km/h$), we used relative parameters, which will allow to calculate the appropriated absolute value for each trajectory. Hence, in our experiments, we used a percentual of the average trajectory speed in order to define the avg and sl parameters. So, $avg = 0.8$ means the avg parameter is 80% of the respective trajectory mean speed. For instance, if a trajectory mean speed is 41,25 km/h, and we use an $avg = 0.8$, then the absolute avg parameter for this trajectory represents a speed of 33 km/h. The same principle is used for the sl parameter.

The minimum time duration ($MinTime$) indicates the minimum time necessary to generate a cluster. It is calculated by subtracting the timestamp of the first point in the cluster from the last point's timestamp in the same cluster [1]. Figure 5.1 shows 4 distinct configurations fixing the parameters avg and sl and varying the parameter $MinTime$. The avg parameter was set to 0.8 and sl was set to 1.0. In 5.1(a) the $MinTime$ value is 90 seconds, in 5.1(b) is 120 seconds, in 5.1(c) 150 seconds and 180 seconds in 5.1(d). As 5.1(a) represents the minimum time value among them, that figure shows the largest number of clusters, five. Little values in this parameter represents more flexibility in order to find clusters. In 5.1(b) and 5.1(c) the number of clusters decreased when $MinTime$ is increased, which is the intuitive behavior of this parameter. However, in 5.1(d) it was found one more cluster than in 5.1(c). This occurred because the $MinTime$ parameter is used in the first step of the clustering. In that step, the limits of points which are looked for in order to be put in the connected-neighborhood is defined by $MinTime$. When that parameter was set to a greater value, the neighborhood becomes greater and the possibility of passing through high speed points and reach a lower speed zone changes, changing the average speed of the cluster. In 5.1(d) the two last clusters of figure 5.1(a) become only one.

The most important parameter is avg. This parameter indicates the percentual of the average speed of the trajectory which will be used as threshold in order to delimit a cluster. Figure 5.2 shows four distinct configurations of this parameter. The $MinTime$ parameter was set to 30s in order to unaffect the generations of clusters. The sl parameter was set to 1 in order to affect as less as possible the evaluation of the avg parameter. In 5.2(a) the avg value is 0.5, in 5.2(b) is 0.6, in 5.2(c) is 0.7 and 0.8 in 5.2(d). The avg parameter has a strict behavior, since the number of clusters increases as greater is its value. This behavior may be seen at Figure 5.2.

At the beginning of CB-SMoT's development, only two parameters were used, *MinTime* and *avg* (cluster average speed). However, these parameters were not enough to generate good clusters in trajectories, since there were some cases where high-speed points were added to a cluster. When a very low-speed sub-trajectory occurs it will decrease significantly its average speed, so in the next moment, when we try to expand the cluster, high-speed neighbors might be added (since this addition keeps the average cluster speed less than the provided by the avg parameter) and the quality of the cluster might be reduced. Figure 5.3 exemplifies this case. In 5.3(a) and 5.3(c) the *sl* parameter is used and in 5.3(b) and 5.3(d) this parameter is not used[2]. As can be seen in figure 5.3, (a) and (c) keep the cluster centered in the low speed points, and in (b) and (d) the cluster tends to aggregate higher speed points. In 5.3(d) the addition of such points affects significantly the quality of the cluster.

With the creation of the parameter sl (Speed Limit), which indicates the highest speed allowed of a point in order to add it to the cluster, we obtained a better quality in the clusters generated using it than those that do not use the parameter. This parameter is used only at the expansion process, so a point whose speed is greater than the speed provided by the sl parameter might be in the set returned by the slowest-neighbor function. If we had not used the sl parameter it would create a cluster with lower quality. The creation of the sl parameter limits the expansion and the addition of non-desirable points.

[1] the cluster ordered by timestamp

[2] 5.3(a) and 5.3(b) *avg* parameter is set to 0.7 and $MinTime$ is set to 30s, in 5.3(a) the sl parameter is set to 0.9; 5.3(c) and 5.3(d) avg parameter is set to 0.8 and $MinTime$ is 30s, in 5.3(c) the sl parameter is 1.0. In 5.3(b) and (d) the sl parameter are not used

Figure 5.4 show four distinct configurations in the parameter sl. The parameters $MinTime$ and avg are fixed as 30 seconds and 0.7, respectively. The parameter sl is a moderator parameter and its creation was used in order to avoid the effect seen at figure 5.3. Looking at the central cluster of figure 5.4 it is possible to better see its effect. It acts avoiding the inclusion of points of high speed to the cluster. In 5.4(a), the sl was set to 0.8, 5.4(b) is set to 0.9, 5.4(c) to 1.0 and 1.1 in 5.4(d). So, the greater the sl parameter is, the faster can be the points be added to the clusters.

5.2 Clustering Experiments

After selecting a limited subset of trajectories that pass through Copacabana, a district in Rio de Janeiro city, we used part of this scenario in order to evaluate our experiments. The whole dataset contains trajectories of sensor cars in Rio de Janeiro at different periods of time. The whole set consists of around 7 million points and 2 thousand trajectories. The subset used consists of 31 trajectories that took place in Copacabana.

Figure 5.5 shows four different trajectories taken from different time periods and with distinct speed. Figure 5.5(a) shows a trajectory that occurred in the period from 04:00 PM to 04:10 PM, (b) occurred in the period from 05:46 PM to 06:00 PM, (c) in the period from 09:02 AM to 09:14 AM, and (d) from 08:04 AM to 08:10 AM.

Over these trajectories we applied the method CB-SMoT [3]. As can be seen in Figure 5.5, the number, the size and the position of the clusters have a big variation from one figure to another. Depending on the application, the clusters can be labeled with the name and position of the street where they occur, with the name of the district, etc.

We performed a briefly comparison between SMoT and CB-SMoT, showing the different outputs of both methods applied in a single trajectory.

Figure 5.6 shows the stops found by the SMoT method with minimal time duration parameter = 60 seconds and districts as relevant feature. As the trajectory spends more than 60s to cross each district, every district crossed by the trajectory will be a stop. So, there are 5 stops: Vidigal, Leblon, Ipanema, Copacabana and Botafogo.

Figure 5.7 shows the same trajectory, but with the stops found by the CB-SMoT method with parameters $MinTime$ = 60 seconds, avg = 0.5 and sl = 0.7. In this case we have one stop labeled Vidigal, four stops labeled Copacabana and two stops labeled Botafogo.

[3] The parameters used to extract these pictures were avg:0.5; MT:60s; sl:0.7

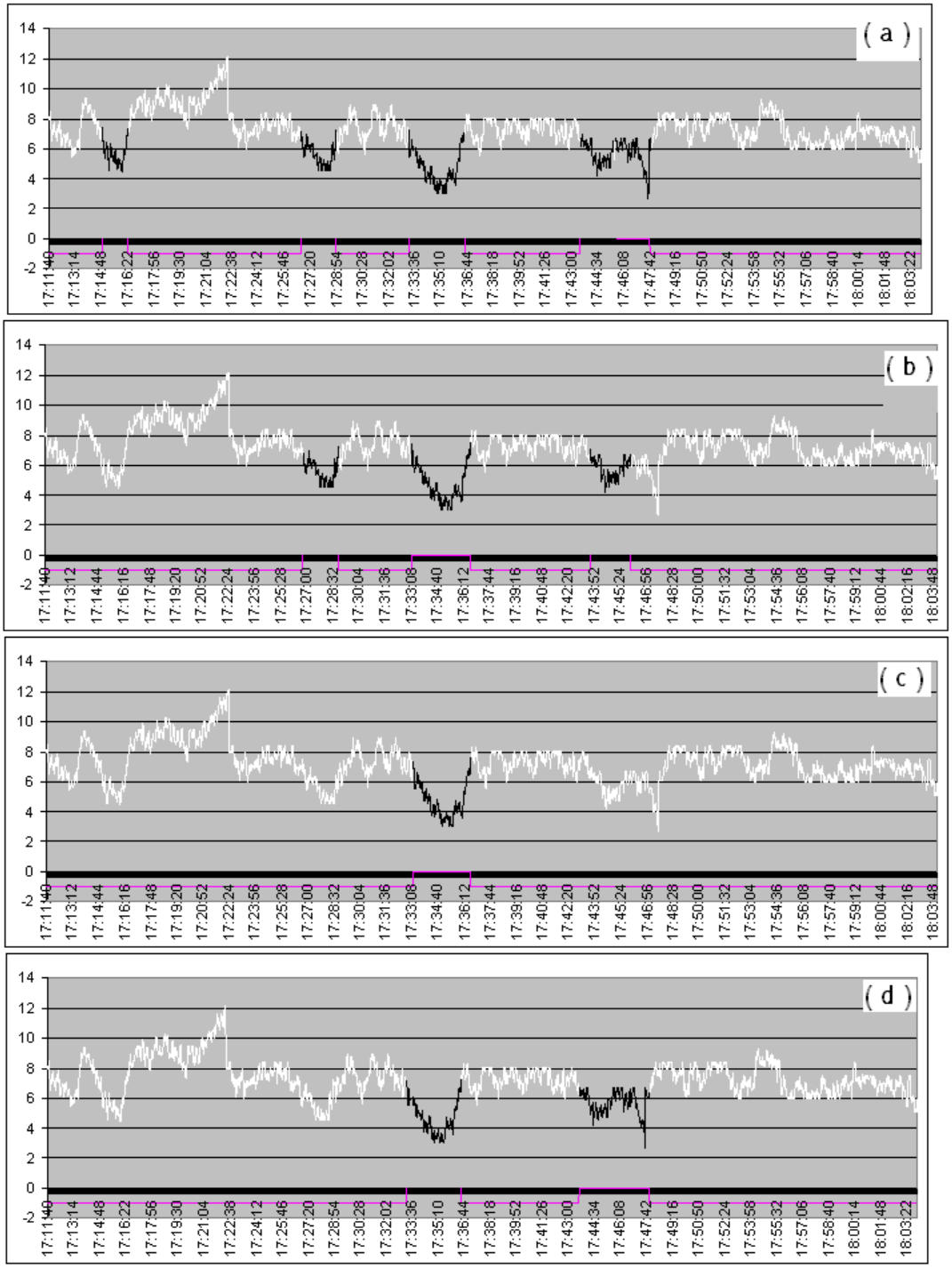

Figure 5.1: Trajectory clusters (in black) using different configurations for the $MinTime$ parameter

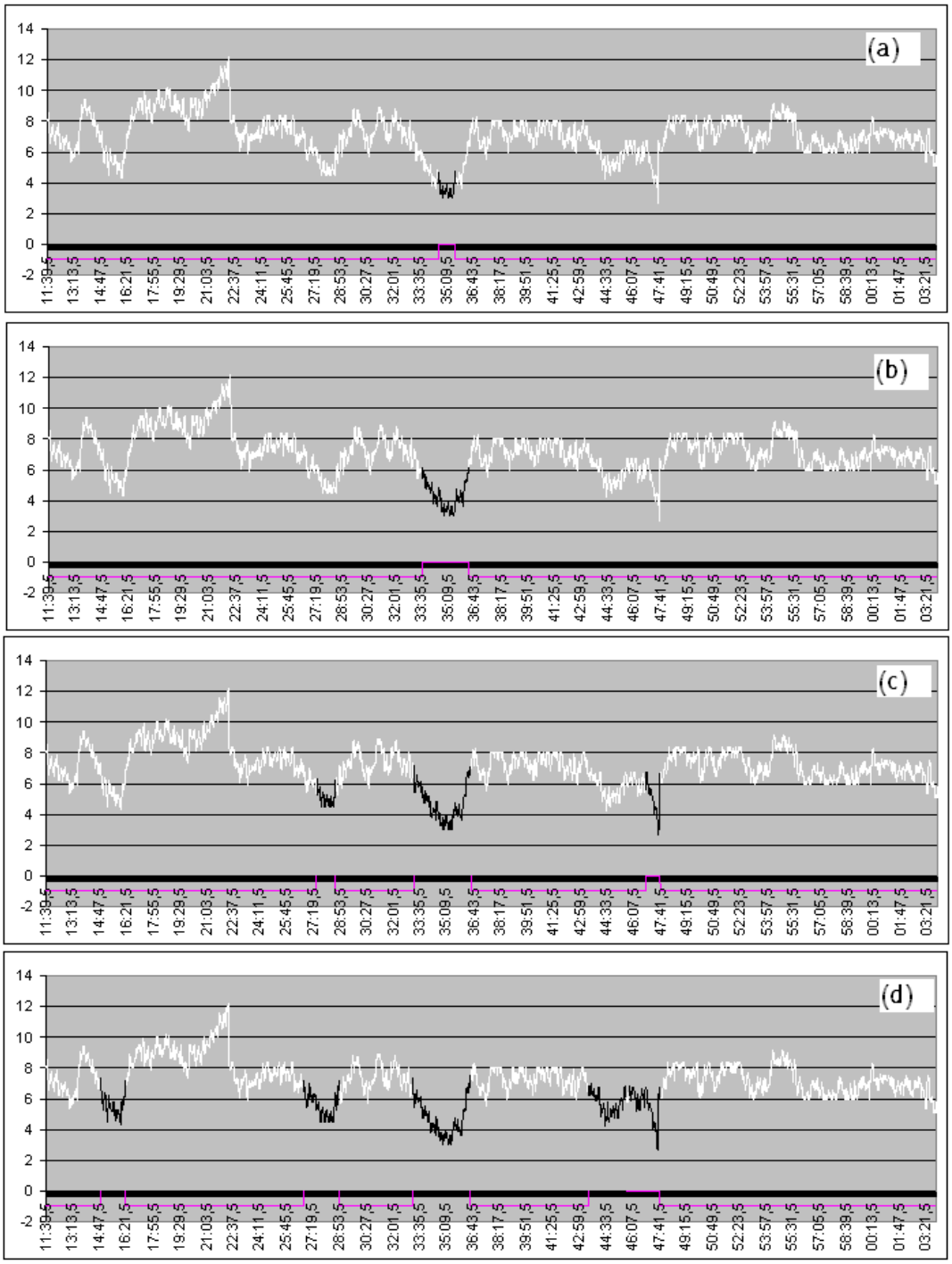

Figure 5.2: Trajectory clusters (in black) using different configurations for the avg parameter

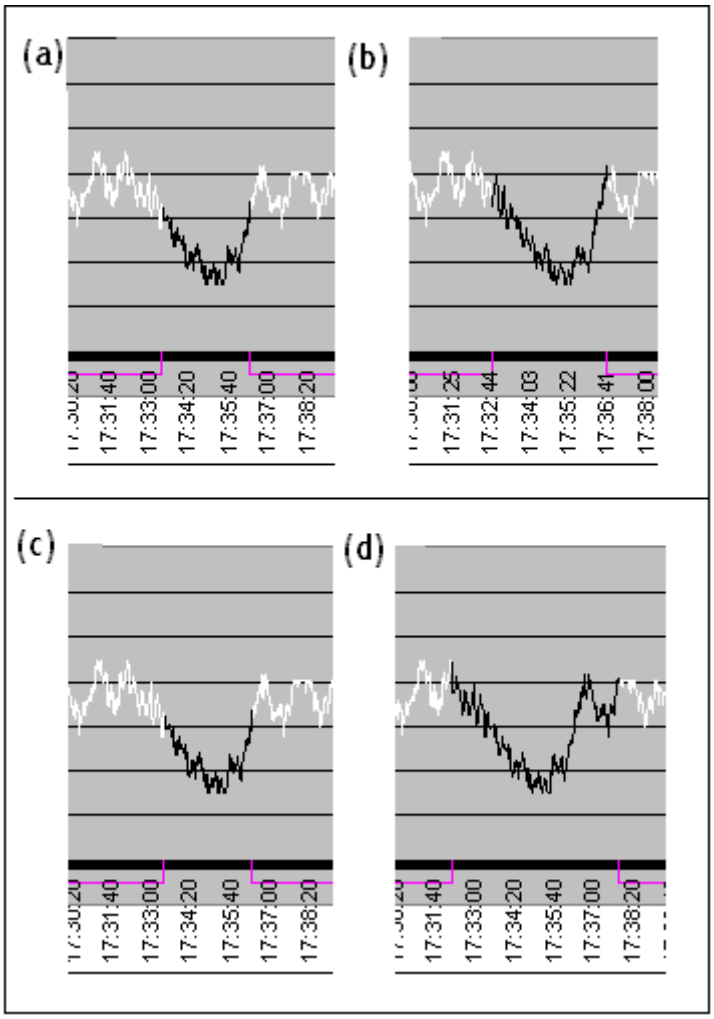

Figure 5.3: Effect of the inclusion of the *sl* parameter in the clusterization step of CB-SMoT. (b) and (d) do not use the *sl* parameter

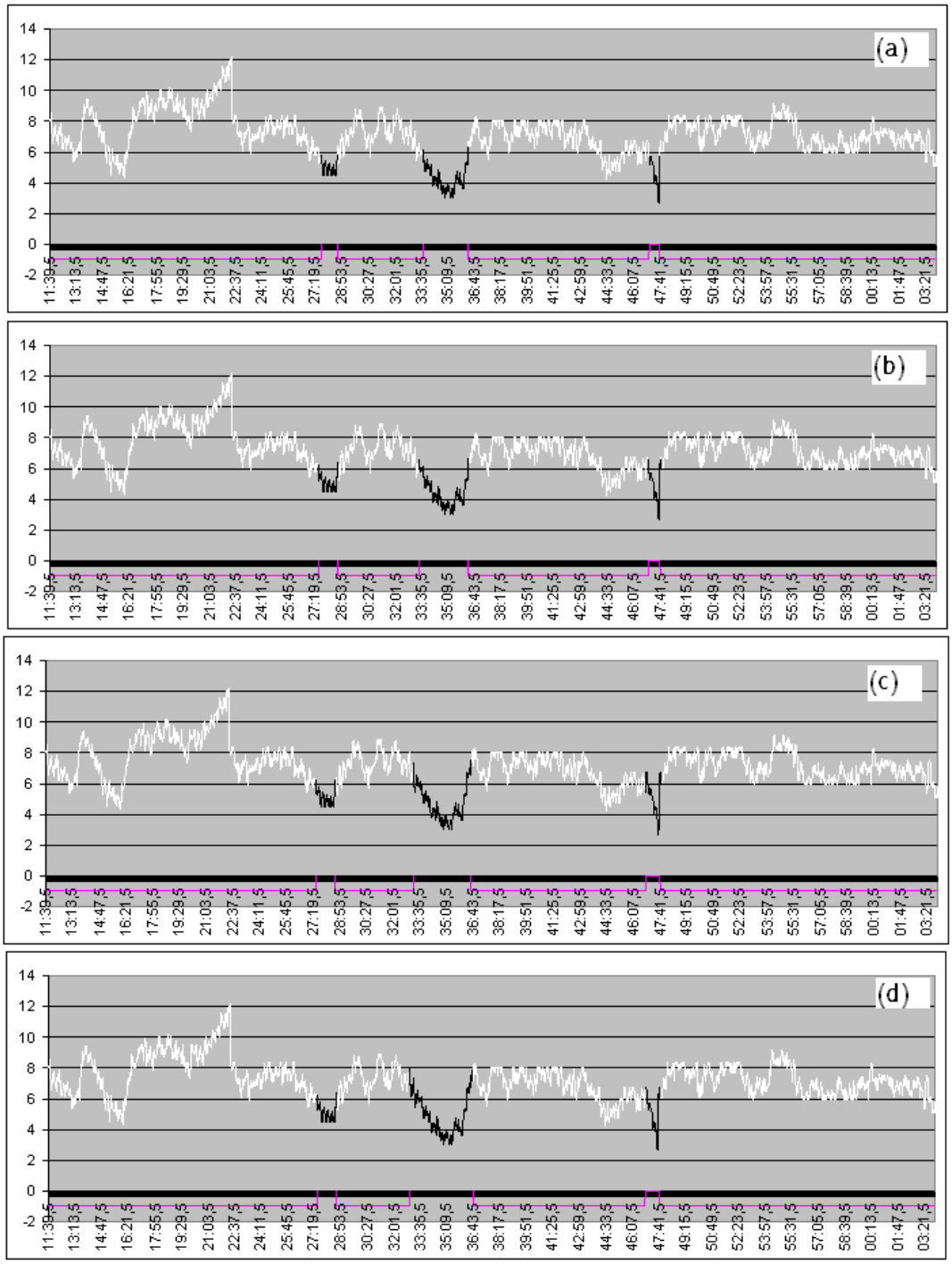

Figure 5.4: Trajectory clusters (in black) using different configurations for the *sl* parameter

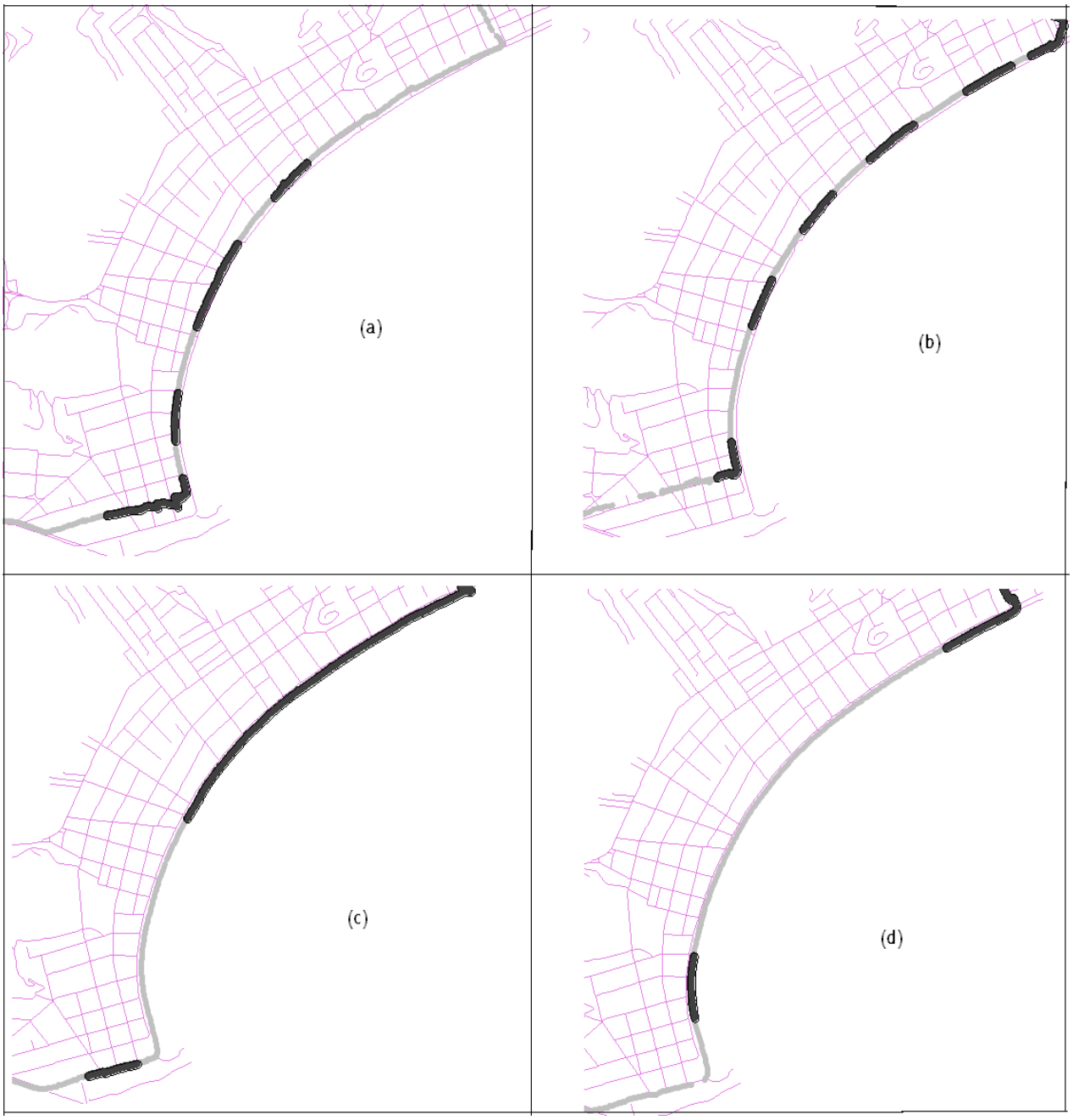

Figure 5.5: Different trajectories and respective clusters in the same region

Figure 5.6: Stops computed by the method SMoT for a single trajectory

Figure 5.7: Stops generated by the method CB-SMoT for a single trajectory

6 CONCLUSION

The increasing amount of location based data, specifically trajectory data, emerged the necessity of intelligent analysis and knowledge discovery from these data. The analysis of trajectory sample points is very limited for several applications. Recently, a new approach based on the notion of interesting places of trajectories (stops) has emerged as a promising way for more meaningful analysis on trajectories.

The usual approach to work with raw trajectory data is separated from any context, and therefore it is impossible to discover semantic patterns. In order to give semantics to these data, a method called SMoT (ALVARES et al., 2007) has been developed to give semantics according to an application domain. The application context is made up of relevant features, and the user may choose which relevant features he is interested in. A relevant feature is any spatial object that is relevant for an application, like a shopping center, a touristic place, ans so on. The SMoT technique is based on the Spaccapietra's (SPACCAPIETRA et al., 2008) *stops* and *moves* approach. Briefly, it is a way to give semantics separating the trajectory in two semantic aspects of interest, *stops* and *moves*. The first one is related to some intrinsic geographic aspect where the object being tracked is considered stopped. For instance, in a bird migration scenario a *stop* could be when the birds stop flying in order to feed (they are fixed in some place). The other concept, *move*, is related to transitions between the *stops*. The SMoT method was the first algorithm developed to give semantics to trajectories given a set of relevant features, but the speed is not taken into account in this method.

In this dissertation we have introduced a new approach to discover interesting places in trajectories. It is a speed-based method to find clusters in single trajectories. In general, the main contributions of this dissertation include:

- A new spatio-temporal clustering algorithm, which is inspired in DBSCAN, where the distance between points is calculated along over the trajectory, instead of the traditional euclidean distance. We consider the notion of minimal time instead of minimal number of points for a region to be considered dense;
- The discovery of interesting places which are not known a priory by the user (unknown stops);
- The generation of stops only in regions where the speed of the trajectory is low;

The goal of this dissertation was to develop a method able to give semantics to trajectories in application domains where low speed needs to be considered. Thus, we created the CB-SMoT (PALMA et al., 2008) method (Clustering-Based SMoT), which uses clustering techniques in order to discover the low speed parts of a single trajectory (clusters).

Obviously, this intrinsic semantics limits the method to application domains where speed is important.

CB-SMoT looks for similar speed points in a spatio-temporal context in order to aggregate them into a cluster, which is given additional semantics in a posterior step in a similar way of SMoT, when the clusters are matched against the background geography. Two kinds of stops can be created: *Known stops* and *Unknown stops*. The *Known stops* are the low speed parts of the trajectory that have a direct association with the background geography. On the other hand, the *Unknown stops* are also low speed parts of the trajectory, but the association with the background geography is not directly related to the important features defined by the user in his application.

Additionally to the new method presented in this dissertation for extracting interesting places from trajectories, we implemented both methods SMoT and CB-SMoT in Weka, with a friendly GUI for trajectory data analysis and knowledge discovery.

In future works, a wide range of possibilities becomes possible in applications related to traditional data mining and a new recent approach associated with sequential data mining. The semantic approach of trajectories is becoming an important data source for novel research, mainly in the knowledge discovery field. Using classical data mining techniques it is possible to find semantic patterns in a set of semantic trajectories. For instance, discover *move* sequences (transition sequences) which may be useful for public transport planning or many other kinds of application. In general, the knowledge extracted will be dependent on the objectives of the application and could be a list of places interesting to the scenario or the most used route of a determinated profile of people. Using a *stop* approach, for example, in a tourism application where the tourists are equipped with GPS devices, so, in their routes, algorithms like SMoT and CB-SMoT would indicate the probable touristic points of the region visited. Additionally, it may be interesting to perform knowledge discovery over the places found out by those methods and the moving relations among them. Another field that may take advantage of this approach is any information system designed to support social or environmental analysis or decision-making (MARK et al., 1999; THERIAULT et al., 2002), like environmental health sciences, where the *lifelines* of individuals are analysed in order to find possible causes of diseases related to the spatio-temporal background.

A more direct future work would be investigate how to incorporate other kinds of semantics into trajectories, besides the geographical one (spatial semantics) and the low speed (spatio-temporal semantics), e.g. provided by CB-SMoT. For instance, other spatio-temporal semantics that might be used would be the low/high acceleration of a trajectory.

The discovery of clusters in regions where sample points are missing is theoretically supported, like, for instance, when a moving object enters in a building and the signal is lost. However, we did not perform direct experiments in order to show that, so it worth to do it as future work.

Another aspect which would be interesting to improve is related to how to give the better label to intersecting *Unknown Stops*. In this dissertation we opted to give the minor *Unknown* identification of the already labeled intersecting *Unknowns*. However this approach is quite simple, and more sophisticated methods might be used.

REFERENCES

ALVARES, L. O.; BOGORNY, V.; KUIJPERS, B.; MACEDO, J. A. F. de; MOELANS, B.; PALMA, A. T. **Towards Semantic Trajectory Knowledge Discovery**. [S.l.]: Hasselt University, 2007.

ALVARES, L. O.; BOGORNY, V.; KUIJPERS, B.; MACEDO, J. A. F. de; MOELANS, B.; VAISAMA, A. A Model for Enriching Trajectories with Semantic Geographical Information. In: ACMGIS, 2007. **Proceedings...** [S.l.: s.n.], 2007. p.162–169.

ANKERST, M.; BREUNIG, M. M.; KRIEGEL, H.-P.; SANDER, J. OPTICS: ordering points to identify the clustering structure. In: SIGMOD 1999, PROCEEDINGS ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, JUNE 1-3, 1999, PHILADEPHIA, PENNSYLVANIA, USA, 1999. **Proceedings...** ACM Press, 1999. p.49–60.

BECKMANN, N.; KRIEGEL, H.-P.; SCHNEIDER, R.; SEEGER, B. The R*-Tree: an efficient and robust access method for points and rectangles. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, ATLANTIC CITY, NJ, MAY 23-25, 1990., 1990. **Proceedings...** ACM Press, 1990. p.322–331.

BIRANT, D.; KUT, A. ST-DBSCAN: an algorithm for clustering spatial-temporal data. **Data Knowl. Eng.**, Amsterdam, The Netherlands, The Netherlands, v.60, n.1, p.208–221, 2007.

BOGORNY, V.; KUIJPERS, B.; ALVARES, L. O. ST-DMQL: a semantic trajectory data mining query language. **International Journal of Geographical Information Science**, [S.l.], 2008.

BOGORNY, V.; PALMA, A. T.; ENGEL, P.; ALVARES, L. O. Weka-GDPM: integrating classical data mining toolkit to geographic information systems. In: WAAMD, 2006. **Proceedings...** SBC, 2006. p.9–16.

BOGORNY, V.; PALMA, A. T.; KUIJPERS, B.; ALVARES, L. O. Spatial Data Mining: from theory to practice with free software. In: WSL INTERNATIONAL WORSHOP OF FREE SOFTWARE, 2007. **Proceedings...** Nova Prova Gráfica e Editora, 2007. p.205–212.

BRAKATSOULAS, S.; PFOSER, D.; TRYFONA, N. Modeling, Storing, and Mining Moving Object Databases. In: IDEAS, 2004. **Proceedings...** [S.l.: s.n.], 2004. p.68–77.

DEMPSTER, A. P.; LAIRD, N. M.; RUBIN, D. B. Maximum Likelihood from Incomplete Data via the EM Algorithm. **Journal of the Royal Statistical Society. Series B (Methodological)**, [S.l.], v.39, n.1, p.1–38, 1977.

DUBES, R. C.; JAIN, A. K. Clustering techniques: the user's dilemma. **Pattern Recognition**, [S.l.], v.8, n.4, p.247–260, 1976.

ESTER, M.; KRIEGEL, H.-P.; SANDER, J.; XU, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: SECOND INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, 1996. **Proceedings...** AAAI Press, 1996. p.226–231.

ESTIVILL-CASTRO, V.; LEE, I. **AMOEBA**: hierarchical clustering based on spatial proximity using Delaunay triangulation. Callaghan 2308, Australia: [s.n.], 1999. (99-05).

FEIGENBAUM, E. A. The simulation of verbal learning behavior. **Computers and Thought**, [S.l.], p.297Ű309, 1963.

FEIGENBAUM EDWARD A., S. H. A. A theory of the serial position effect. **Br. J. Psychol.**, [S.l.], v.53, p.307Ű320, 1962.

FEIGENBAUM EDWARD A., S. H. A. EPAM-like Models of Recognition and Learning. **Cognitive Science: A Multidisciplinary Journal**, [S.l.], v.8, p.306–336, 1984.

FERNAND GOBET PETER C. R. LANE, S. C. P. C.-H. C. G. J. I. O.; PINE, J. M. Chunking mechanisms in human learning. **Trends in Cognitive Science**, [S.l.], v.5, n.6, p.236–243, 2001.

FISHER, D. H. Knowledge Acquisition via Incremental Conceptual Clustering. **Machine Learning**, [S.l.], v.2, n.2, p.139–172, 1987.

FISHER, L.; NESS, J. W. V. Admissible clustering procedures. **Biometrika**, [S.l.], v.58, n.1, p.91–104, 1971.

GENNARI, J. H.; LANGLEY, P.; FISHER, D. H. Models of Incremental Concept Formation. **Artif. Intell.**, [S.l.], v.40, n.1-3, p.11–61, 1989.

GüTING, R. H.; ALMEIDA, V. T. D.; DING, Z. Modeling and querying moving objects in networks. **VLDB J**, [S.l.], v.15, p.2006, 2004.

GÜTING, R. H.; BÖHLEN, M. H.; ERWIG, M.; JENSEN, C. S.; LORENTZOS, N. A.; SCHNEIDER, M.; VAZIRGIANNIS, M. A foundation for representing and quering moving objects. **ACM Trans. Database Syst.**, [S.l.], v.25, n.1, p.1–42, 2000.

GÜTING, R. H.; SCHNEIDER, M. **Moving Objects Databases**. [S.l.]: Morgan Kaufmann, 2005.

IBA, W. Learning to Classify Observed Motor Behavior. In: IJCAI, 1991. **Proceedings...** [S.l.: s.n.], 1991. p.732–738.

IBA, W.; LANGLEY, P. Unsupervised Learning of Probabilistic Concept Hierarchies. **Lecture Notes in Computer Science**, [S.l.], v.2049, p.39–70, 2001.

INTERNATIONAL CONFERENCE ON MOBILE DATA MANAGEMENT (MDM 2007), MANNHEIM, GERMANY, MAY 7-11, 2007, 8., 2007. **Proceedings...** IEEE, 2007.

JAIN, A. K.; DUBES, R. C. **Algorithms for clustering data**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

JAIN, A. K.; MURTY, M. N.; FLYNN, P. J. Data clustering: a review. **ACM Comput. Surv.**, New York, NY, USA, v.31, n.3, p.264–323, 1999.

KING, B. Step-wise clustering procedures. **Journal of the American Statistical Association**, [S.l.], v.69, p.86–101, 1967.

KUIJPERS, B.; OTHMAN, W. Trajectory Databases: data models, uncertainty and complete query languages. In: ICDT, 2007. **Proceedings...** [S.l.: s.n.], 2007. p.224–238.

LAUBE, P.; IMFELD, S.; WEIBEL, R. Discovering relative motion patterns in groups of moving point objects. **International Journal of Geographical Information Science**, [S.l.], v.19, n.6, p.639–668, 2005.

LEBOWITZ, M. Experiments with Incremental Concept Formation: unimem. **Machine Learning**, [S.l.], v.2, n.2, p.103–138, 1987.

LEE, J.-G.; HAN, J.; WHANG, K.-Y. Trajectory clustering: a partition-and-group framework. In: SIGMOD CONFERENCE, 2007. **Proceedings...** ACM, 2007. p.593–604.

LU, S.-Y.; FU, K. S. A Sentence-to-Sentence Clustering Procedure for Pattern Analysis. **Systems, Man and Cybernetics, IEEE Transactions on**, [S.l.], v.8, n.5, p.381–389, May 1978.

MA, D.; ZHANG, A. An Adaptive Density-Based Clustering Algorithm for Spatial Database with Noise. **icdm**, Los Alamitos, CA, USA, v.00, p.467–470, 2004.

MACQUEEN, J. B. Some Methods for Classification and Analysis of MultiVariate Observations. In: BERKELEY SYMPOSIUM ON MATHEMATICAL STATISTICS AND PROBABILITY, 1967. **Proceedings...** University of California Press, 1967. v.1, p.281–297.

MARK, D.; EGENHOFER, M.; BIAN, L.; HORNSBY, K.; ROGERSON, P.; VENA, J. **Spatio-temporal GIS analysis for environmental health using geospatial lifelines**. 1999.

MITCHELL, T. **Machine Learning**. [S.l.]: McGraw-Hill Education (ISE Editions), 1997.

MURTAGH, F. A Survey of Recent Advances in Hierarchical Clustering Algorithms. **Comput. J.**, [S.l.], v.26, n.4, p.354–359, 1983.

NANNI, M.; PEDRESCHI, D. Time-focused clustering of trajectories of moving objects. **Journal of Intelligent Information Systems**, [S.l.], v.27, n.3, p.267–289, November 2006.

NG, R. T.; HAN, J. Efficient and Effective Clustering Methods for Spatial Data Mining. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, SEPTEMBER 12–15, 1994, SANTIAGO, CHILE PROCEEDINGS, 20., 1994, Los Altos, CA 94022, USA. **Proceedings...** Morgan Kaufmann Publishers, 1994. p.144–155.

PALMA, A. T.; BOGORNY, V.; KUIJPERS, B.; ALVARES, L. O. A clustering-based approach for discovering interesting places in trajectories. In: SAC '08: PROCEEDINGS OF THE 2008 ACM SYMPOSIUM ON APPLIED COMPUTING, 2008, New York, NY, USA. **Proceedings...** ACM, 2008. p.863–868.

SANDER, J.; ESTER, M.; KRIEGEL, H.-P.; XU, X. Density-Based Clustering in Spatial Databases: the algorithm gdbscan and its applications. **Data Min. Knowl. Discov.**, Hingham, MA, USA, v.2, n.2, p.169–194, 1998.

SNEATH, P. H. A.; SOKAL, R. R. **Numerical Taxonomy**. [S.l.]: Freeman, 1973.

SPACCAPIETRA, S.; PARENT, C.; DAMIANI, M. L.; MACEDO, J. A. de; PORTO, F.; VANGENOT, C. A conceptual view on trajectories. **Data Knowl. Eng.**, Amsterdam, The Netherlands, The Netherlands, v.65, n.1, p.126–146, 2008.

SPACCAPIETRA, S.; PARENT, C.; DAMIANI, M.-L.; MACEDO, J. A. F. de; PORTO, F.; VANGENOT, C. **A Conceptual View on Trajectories**. [S.l.]: Ecole Polytechnique Federal de Lausanne, 2007.

THERIAULT, M.; CLARAMUNT, C.; SEGUIN, A.; VILLENEUVE, P. Temporal GIS and Statistical Modeling of Personal Lifelines. In: ADVANCES IN SPATIAL DATA HANDLING, 2002. **Proceedings...** Springer-Verlag, 2002. p.433–449.

THOMPSON, K.; LANGLEY, P. **Concept formation in structured domains**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991. 127–161p.

WARD, J. H. Hierarchical Grouping to Optimize an Objective Function. **Journal of the American Statistical Association**, [S.l.], v.58, n.301, p.236–244, 1963.

WITTEN, I. H.; FRANK, E. **Data Mining**: practical machine learning tools and techniques. 2.ed. [S.l.]: Morgan Kaufmann, 2005. (Morgan Kaufmann Series in Data Management Systems).

WOLFSON, O.; XU, B.; CHAMBERLAIN, S.; JIANG, L. Moving Objects Databases: issues and solutions. In: STATISTICAL AND SCIENTIFIC DATABASE MANAGEMENT, 1998. **Proceedings...** [S.l.: s.n.], 1998. p.111–122.

ZAHN, C. Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters. **Computers, IEEE Transactions on**, [S.l.], v.C-20, n.1, p.68–86, Jan. 1971.

ZHOU, C.; FRANKOWSKI, D.; LUDFORD, P. J.; SHEKHAR, S.; TERVEEN, L. G. Discovering personally meaningful places: an interactive clustering approach. **ACM Trans. Inf. Syst.**, [S.l.], v.25, n.3, 2007.

ZHOU, C.; FRANKOWSKI, D.; LUDFORD, P.; SHEKHAR, S.; TERVEEN, L. **Discovering personal gazetteers**: an interactive clustering approach. 2004.

APPENDIX A UMA ABORDAGEM BASEADA EM CLUSTERIZAÇÃO PARA A DESCOBERTA DE LUGARES DE INTERESSE EM TRAJETÓRIAS

A crescente quantidade de dados provenientes de dispositivos móveis, sejam eles de localização, tais como GPS, ou comunicação, como celulares, acarreta um potencial imenso de dados possíveis de ser usados em várias áreas da informática. Principalmente no campo de descoberta de conhecimento (data mining), onde esse novo tipo de dado, trajetórias de objetos móveis, possibilita uma gama enorme de possíveis aplicações. Essa perspectiva demanda novos métodos especializados para serem aplicados a fim de se conseguir obter um nível mais alto de conhecimento.

Em vista desse novo campo, novos modelos conceituais e algoritmos estão sendo criados. Em um desses modelos conceituais (SPACCAPIETRA et al., 2008), os pontos das trajetória são divididos em trechos de interesse, chamados *stops e moves*. No trabalho dessa dissertação exploramos uma característica que achamos ser intuitiva nos *stops*, a baixa velocidade, e propusemos um algoritmo que procura trechos de baixa velocidade na trajetória a fim de nomeá-los como *stops* e mais tarde atribuir outros tipos de semântica.

No Capítulo 2 são apresentados os principais trabalhos relacionados à pesquisa desenvolvida. São apresentadas a intuição do que é uma trajetória e a definição que é utilizada no restante do trabalho. As trajetórias passam de um nível mais bruto (formado apenas pelos dados na forma (x,y,t) provenientes dos dispositivos) para um nível de abstração mais elevado (composta de *stops* e *moves*).

A atribuição de semântica às trajetórias é um campo novo na informática e teve seu começo com o modelo de *stops* e *moves*. Esses novos conceitos particionam a trajetória em segmentos que são um dos dois conceitos expostos. Os *stops*, de acordo com Spaccapietra, são trechos de interesse naquela trajetória. Entretanto Spaccapietra não define como a semântica é atribuída a esses trechos/conceitos.O SMoT (Stops and Moves of Trajectories) (ALVARES et al., 2007) foi o primeiro algoritmo definido para gerar automaticamente stops e moves. O SMoT é baseado na atribuição de semântica considerando a geografia por onde passa a trajetória. Ele usa a geografia a fim de identificar os aspectos geográficos que estão relacionados com os *stops* e, dessa forma, atribuir uma semântica aos mesmos.

Para a atribuição de semântica pelo SMoT faz-se necessário a definição de alguns conceitos básicos, entre eles o *candidate stop*. O *candidate stop* é uma tupla (R_c, Δ_c), onde R_c define uma área que representa a geometria de um objeto geográfico, e Δ_c representa a quantidade de tempo mínima de intersecção com uma trajetória para que o objeto seja considerado de interesse (stops). Para que um *candidate stop* venha a ser um *stop*, é necessário que:

1. A trajetória tenha intersecção com o R_c do *candidate stop*

2. O tempo de intersecção contínua com o R_c deve ser maior que o Δ_c do *candidate stop*

Os dois itens combinados dizem que se uma trajetória intercepta um *candidade stop* em algum momento então o tempo de intersecção deve ser maior que o Δ_c do *candidate stop*. Adicionalmente todos os pontos desse trecho devem obrigatoriamente interceptar o mesmo R_c do candidate stop. Dessa forma, o *candidate stop* passa a ser um *stop* para aquela trajetória.

Para explorar o uso da baixa velocidade e descobrir os trechos lentos, fez-se o uso de técnicas de clusterização. Dessa forma a maioria dos trabalhos relacionados são algoritmos desse campo. É dada uma ênfase maior para o DBSCAN (ESTER et al., 1996), que é o trabalho que mais influenciou a proposta dessa pesquisa. DBSCAN é um algoritmo de clusterização de pontos baseado em densidade que usa a quantidade de pontos como base para a geração de clusters. Quando a densidade de determinado ponto satisfazer alguns requisitos, como, por exemplo, a quantidade mínima de pontos, a área vizinha é expandida, agrupando-se outros pontos chaves. Dessa forma é possível obter clusters das mais variadas formas.

O método CB-SMoT (Clustering-Based SMoT), proposto nessa dissertação, no Capítulo 3, tem duas etapas principais: clusterização e atribuição de semântica. Na parte de clusterização, CB-SMoT foi inspirado pelo algoritmo baseado em densidade DBSCAN. Usando conceitos espaço-temporais ao invés de apenas a abordagem clássica de densidade para a criação de clusters, CB-SMoT considera adicionalmente o tempo no processo de clusterização, de forma que os clusters correspondem a trechos em que a velocidade é lenta.

Com o desenvolvimento do método CB-SMoT as trajetórias não são analisadas com base apenas na geografia e contexto que existem no espaço físico em que ocorrem. Outra característica é levada em conta: sua velocidade. Dessa forma temos dois aspectos envolvidos com a trajetória: o primeiro é relacionado apenas com a trajetória, que é sua velocidade; e o segundo é o contexto espacial que existe por detrás da trajetória, a geografia. Cada aspecto é tratado de uma maneira independente, o primeiro utiliza clusterização para definir os aspectos de interesse na trajetória, que no caso do CB-SMoT são os trechos de menor velocidade. O segundo aspecto usa a geografia de fundo, de uma maneira similar ao SMoT para uma atribuição de semântica. Este aspecto relaciona a trajetória com um contexto que depende dos *candidate stops* de interesse do usuário.

A semântica atribuída pelo CB-SMoT é dada em cada um desses passos. No passo de clusterização, com a descoberta dos trechos lentos da trajetória, a primeira questão semântica é envolvida, já que se atribui uma semântica de baixa velocidade. Esse primeiro passo diz respeito apenas a trajetória e não considera o espaço onde a trajetória está inserida. Adicionalmente ele é um filtro das partes importantes que devem ser levadas em conta no segundo passo. Inicialmente é obtida uma vizinhança com duração mínima igual ao parâmetro *MinTime*. Essa vizinhança parte de um ponto e os pontos vizinhos mais lentos vão sendo adicionados até que se atinja o tempo mínimo requerido. Depois de obtida a vizinhança de tempo *MinTime*, chamada de *slowest-neighborhood*, verificamos se a velocidade média dela é inferior ao parâmetro *avg*, se for ela é considerada uma vizinhança núcleo (*core-neighborhood*). As vizinhanças núcleo são expandidas em um segundo momento, tentando-se agregar outros pontos de baixa velocidade. Dessa forma, vão sendo adicionados os pontos vizinhos que tenham velocidade inferior ao parâmetro *sl* e que

mantenham a velocidade média total da vizinhança abaixo do parâmetro *avg*. Terminada a expansão da *slowest-neighborhood* chama-se essa nova vizinhança de *connected-neighborhood* (vizinhança conectada) e o próximo passo é escolher as vizinhanças com menor velocidade para se tornarem clusters. Quando duas ou mais vizinhanças conectadas compartilham ao menos um ponto em comum, torna-se um cluster aquela que tiver a menor velocidade média. Na implementação do algoritmo começa-se analisando os pontos de menor velocidade de forma que o processo de descoberta de clusters fica otimizado.

O segundo passo, a atribuição de semântica geográfica, diz respeito ao contexto geográfico da trajetória. Nesse momento os trechos de interesse provenientes do passo anterior são analisados e procura-se agregar novas informações semânticas baseadas em seu contexto espacial. Dessa forma pode-se ter dois tipos de *stops*: *known stops* e os *unknown stops*. Os primeiros estão relacionados a pontos de interesse do usuário que existem na base de dados de *candidate stops*. Os *unknown stops*, por outro lado, são pontos que foram capturados no primeiro passo como trechos com alto potencial semântico (devido a sua baixa velocidade), mas que pela seleção de interesse do usuário ou falta de *candidate stops* na base de dados não está relacionada a nenhuma entidade/evento geográfico diretamente.

No Capítulo 4 é apresentado o protótipo desenvolvido, o módulo STPM (*Semantic Trajectory Preprocessing Module*. Esse módulo foi agregado ao Weka (WITTEN; FRANK, 2005), que é um programa bem conhecido na área de mineração de dados. No módulo STPM o usuário tem a opção de escolher os *candidate stops* que ele está interessado e também escolher as trajetórias em questão. No módulo ele tem a opção de rodar tanto o SMoT quanto o CB-SMoT, que por final vai gerar uma tabela de stops. Essa tabela ainda pode ser usada para criação de arquivos do tipo *arff* (uma parte também implementada no módulo) e assim possibilitar o uso da mineração tradicional de dados, que é parte nativa do Weka.

No Capítulo 5, são apresentados alguns experimentos tanto para a parte de clusterização quanto para a parte de atribuição de semântica. Os experimentos iniciais são realizados com dados sintéticos para enfatizar os efeitos de cada parâmetro na geração dos clusters. Dessa forma varia-se os parametros *avg, MinTime e sl* a fim de se obter um retorno visual da sua influência. O parâmetro *avg* possui um comportamento linear, ou seja, conforme é aumentando seu valor, aumentamos a velocidade média de tolerância do cluster, dessa forma seu aumento tende a um aumento na quantidade de clusters encontrados. O parâmetro *MinTime* diz respeito a duração do cluster, dessa forma, temos um comportamento praticamente linear também, conforme aumentamos seu valor estamos sendo mais criteriosos na duração do cluster e, dessa forma, a quantidade de cluster tende a diminuir. O parâmetro *sl* age na clusterização como um moderador, indicando quais pontos podem ser adicionados ao cluster. Ele age considerando a velocidade dos pontos, de forma que não é permitido adicionar pontos que tenham sua velocidade acima desse limite. O parâmetro *sl* é importante nos casos onde existe um trecho de velocidade muito baixa e de longa duração (fazendo a velocidade média do trecho ser baixa). Nesse caso a média é puxada para baixo e abre-se uma brecha para adicionar pontos de alta velocidade, já que a inclusão desses pontos deixaria a média total ainda abaixo da média limite (parâmetro *avg*) e diminuiria a qualidade do cluster.

Nessa dissertação as principais contribuições estão relacionadas à parte de clusterização do CB-SMoT e a atribuição de semântica. A clusterização é uma nova abordagem espaço-temporal, que utiliza a velocidade como critério de geração do cluster. Outros métodos costumam considerar apenas aspectos espaciais (dados (x,y) dos pontos), como

a grande maioria dos métodos baseados em densidade. Dessa forma, CB-SMoT integra a clusterização em um conceito espaço-temporal de fato, ou seja, a velocidade. No aspecto de atribuição de semântica, aspectos geográficos desconhecidos pelo usuário (*unknown stops*), que são trechos de baixa velocidade mas sem informação geográfica, podem ser descobertos.

Adicionalmente, os *unknown stops* são partes potenciais de interesse pelo usuário, já que são partes lentas da trajetória. Entretanto apenas rotular como possível ponto de interesse não seria muito prático de um ponto de vista mais amplo, como mineração de dados. Dessa forma, procuramos nomear os *unknowns* de acordo com sua proximidade espacial, tentando buscar relações a entidades aparentemente sem nenhuma ligação. Essa abordagem permite em aplicações mais específicas uma maior facilidade de descoberta de padrões, já que rotulamos os *unknowns* de acordo com uma relação geográfica entre eles, e não apenas como simples instâncias independentes uma das outras.

As áreas que mais ganham com o desenvolvimento do CB-SMoT é a de clusterização, que ganha um método totalmente espaço-temporal. Com isso é possível considerar a velocidade em aplicações onde esse aspecto é relevante, como por exemplo, uma aplicação de trânsito, onde trechos de baixa velocidade podem indicar congestionamentos. Dessa forma podemos atribuir semântica a esses trechos, fazendo com que o *stop* em questão seja mais especializado do que se aplicássemos um algoritmo que leva em conta apenas geografia por onde passa a trajetória. Adicionamente, a parte de mineração de dados fica com uma nova ferramenta para a descoberta de padrões, principalmente no campo de padrões sequenciais, que é caracterizada pela descoberta de padrões ao longo do tempo. Como os *stops e moves* são unidades semânticas de grande interesse em trajetórias, eles podem ser usados como entrada nesse promissor campo de pesquisa.