



MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2017/07.06.12.27-TDI

A FRAMEWORK FOR TRAJECTORY DATA MINING

Diego Vilela Monteiro

Master's Dissertation of the
Graduate Course in Applied
Computing, guided by Drs. Karine
Reis Ferreira, and Rafael Duarte
Coelho dos Santos, approved in
June 29, 2017.

URL of the original document:

<<http://urlib.net/8JMKD3MGP3W34P/3P8ANQ2>>

INPE
São José dos Campos
2017

PUBLISHED BY:

Instituto Nacional de Pesquisas Espaciais - INPE

Gabinete do Diretor (GB)

Serviço de Informação e Documentação (SID)

Caixa Postal 515 - CEP 12.245-970

São José dos Campos - SP - Brasil

Tel.:(012) 3208-6923/6921

E-mail: pubtc@inpe.br

**COMMISSION OF BOARD OF PUBLISHING AND PRESERVATION
OF INPE INTELLECTUAL PRODUCTION (DE/DIR-544):****Chairperson:**

Maria do Carmo de Andrade Nono - Conselho de Pós-Graduação (CPG)

Members:

Dr. Plínio Carlos Alvalá - Centro de Ciência do Sistema Terrestre (CST)

Dr. André de Castro Milone - Coordenação de Ciências Espaciais e Atmosféricas (CEA)

Dra. Carina de Barros Melo - Coordenação de Laboratórios Associados (CTE)

Dr. Evandro Marconi Rocco - Coordenação de Engenharia e Tecnologia Espacial (ETE)

Dr. Hermann Johann Heinrich Kux - Coordenação de Observação da Terra (OBT)

Dr. Marley Cavalcante de Lima Moscati - Centro de Previsão de Tempo e Estudos Climáticos (CPT)

Silvia Castro Marcelino - Serviço de Informação e Documentação (SID) **DIGITAL LIBRARY:**

Dr. Gerald Jean Francis Banon

Clayton Martins Pereira - Serviço de Informação e Documentação (SID)

DOCUMENT REVIEW:

Simone Angélica Del Ducca Barbedo - Serviço de Informação e Documentação (SID)

Yolanda Ribeiro da Silva Souza - Serviço de Informação e Documentação (SID)

ELECTRONIC EDITING:

Marcelo de Castro Pazos - Serviço de Informação e Documentação (SID)

André Luis Dias Fernandes - Serviço de Informação e Documentação (SID)



MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2017/07.06.12.27-TDI

A FRAMEWORK FOR TRAJECTORY DATA MINING

Diego Vilela Monteiro

Master's Dissertation of the Graduate Course in Applied Computing, guided by Drs. Karine Reis Ferreira, and Rafael Duarte Coelho dos Santos, approved in June 29, 2017.

URL of the original document:

<<http://urlib.net/8JMKD3MGP3W34P/3P8ANQ2>>

INPE
São José dos Campos
2017

Cataloging in Publication Data

Monteiro, Diego Vilela.

M764f A framework for trajectory data mining / Diego Vilela
Monteiro. – São José dos Campos : INPE, 2017.
xx + 61 p. ; (sid.inpe.br/mtc-m21b/2017/07.06.12.27-TDI)

Dissertation (Master in Applied Computing) – Instituto
Nacional de Pesquisas Espaciais, São José dos Campos, 2017.

Guiding : Drs. Karine Reis Ferreira, and Rafael Duarte Coelho
dos Santos.

1. R. 2. Trajectory. 3. Data-mining. I.Title.

CDU 004.93



Esta obra foi licenciada sob uma Licença [Creative Commons Atribuição-NãoComercial 3.0 Não Adaptada](#).


This work is licensed under a [Creative Commons Attribution-NonCommercial 3.0 Unported License](#).

Aluno (a): **Diego Vilela Monteiro**

Título: **"A FRAMEWORK FOR TRAJECTORY DATA MINING"**

Aprovado (a) pela Banca Examinadora
em cumprimento ao requisito exigido para
obtenção do Título de **Mestre** em
Computação Aplicada

Dra. Lúbia Vinhas


Presidente / INPE / São José dos Campos - SP

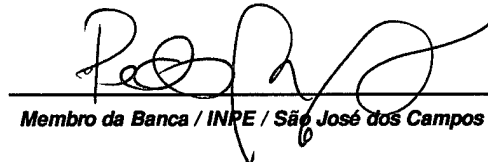
Dra. Karine Reis Ferreira


Orientador(a) / INPE / São José dos Campos - SP


Dr. Rafael Duarte Coelho dos Santos


Orientador(a) / INPE / SJCampos - SP

Dr. Pedro Ribeiro de Andrade Neto


Membro da Banca / INPE / São José dos Campos - SP

Dr. Sandro Klippel


Convidado(a) / IBAMA / Itajaí - SC

Este trabalho foi aprovado por:

() maioria simples

☒ unanimidade

São José dos Campos, 29 de junho de 2017

*“Carry on my wayward son
For there’ll be peace when you are done
Lay your weary head to rest
Don’t you cry no more”*

KANSAS
in *“Leftoverture”*, 1976

To my loved ones, specially those who can't read this anymore.

ACKNOWLEDGEMENTS

First of all I would like to thank my family for all the support and incentive. Without them I would not have gotten here. Without their unconditional love I wouldn't be who I am today. I also thank my friends for putting up with me.

I thank my advisors, Karine and Rafael, for all the help, corrections, and teachings.

I am grateful to Pedro Andrade, for showing me the basis of R.

This work was financially supported by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior).

ABSTRACT

Spatiotemporal data are everywhere, being collected from different devices such as Earth Observation and GPS satellites, sensor networks, vehicles and smartphones. Data collected from those devices may contain valuable information about different subjects, including environmental monitoring, weather as well as mobility. Of these subjects, one of particular interest is moving objects' trajectory data. In order to process this kind of data, there is a need for high-level programming environments that allow users to quickly and easily develop new algorithms. In this work, I propose a framework that extends the R environment for big trajectory data mining. I designed and developed two new packages that allow R users to efficiently deal with big trajectory data sets and fast implement new mining algorithms over them. I also propose an efficient method to discover partners in moving object trajectories. Such method identifies pairs of trajectories whose objects stay together during certain periods, based on distance time series analysis. Finally, I validate both the framework and method via case studies.

Keywords: R. Trajectory. Data-Mining.

UM FRAMEWORK PARA MINERAÇÃO DE BIG DATA DE TRAJETÓRIAS

RESUMO

Dados espaçotemporais estão em todos os lugares, sendo coletados por diversos equipamentos como satélites GPS e de Observação da Terra, redes de sensores, veículos e *smartphones*. Dados coletados por esses equipamentos contêm informações valiosas sobre diversas áreas como monitoramento ambiental, clima assim como mobilidade. Dentro dessas áreas, uma de interesse especial é a de trajetórias de objetos móveis. Para poder processar tais dados, existe a necessidade de um ambiente de alto nível que permite ao usuário desenvolver rapidamente e facilmente novos algoritmos. Neste trabalho é proposto um *framework* para estender o ambiente R para a mineração de grandes bases de dados de trajetória. Projeta-se e desenvolve-se dois novos pacotes que permitem usuários R manipular eficientemente grandes conjuntos de dados de trajetórias e a rápida implementação de novos algoritmos de mineração neles. Propõe-se também um método eficiente para encontrar parceiros em trajetórias de objetos móveis. Tal método identifica pares de trajetórias cujos objetos permanecem juntos durante certos períodos, baseado em análise de series temporais. Finalmente, valida-se tanto o método quanto o *framework* via estudos de caso.

Palavras-chave: R. Trajetória. Mineração de Dados.

LIST OF FIGURES

	<u>Page</u>
2.1 An Overview of the Steps That Compose the KDD Process.	6
2.2 Inconsistent trajectory detected by speed filter.	7
2.3 Exemplification of Douglas-Peucker algorithm for line simplification. . . .	8
2.4 Exemplification of open-window strategy.	8
2.5 Single Trajectory enriched semantically.	9
2.6 Terralib 5 ST module	14
2.7 Model used in part of Terralib 5 architecture.	15
3.1 Discrete Trajectory Representation of 4 moving objects	17
3.2 Trajectories and their STBoxes.	18
3.3 System architecture.	19
4.1 Example of <i>partner</i> trajectories	28
4.2 Example of trajectories which are not <i>partners</i>	29
4.3 Trajectories with three possible <i>partner</i> interpretations	29
4.4 Steps in <i>Partner</i> discovery.	33
5.1 (Above) R Plot of trajectories of sea elephant 40 and (below) Google Earth plot of all sea elephant trajectories.	36
5.2 R script using TrajDataAccess package to select trajectories from a KML file.	36
5.3 (Above) R Plot of trajectories of sea elephants 40 and 41 (below) R Plot of their distance time-series.	37
5.4 All data plotted against the map of Brazil.	38
5.5 Two trajectories from the same vessel.	38
5.6 R script using TrajDataAccess package to select trajectories from a PostGIS database based on a spatiotemporal constraint.	39
5.7 Selected trajectories (in the box) against all trajectories	40
5.8 Code to get STBoxes of adequate size	40
5.9 Beginning of the list of obtained STBoxes	41
5.10 Some parts of the obtained STBoxes (horizontal black lines) against the trajectories	42
5.11 Selected trajectories (in the box) against all trajectories	42
5.12 Original trajectory (above) and Douglas-Peucker compression (below), no visible changes.	43
5.13 Original trajectory (above) open-window Meratnia-By compression (be- low), no visible changes.	44
5.14 Original trajectory (blue) filtered trajectory (red).	45

5.15	Trajectory comparison of part of original trajectory (red crosses) and part of filtered trajectory (clue circles).	46
5.16	Code to find <i>Partners</i> in a DataBase.	47
5.17	Vessels around the Brazilian coast. Blue dots/lines: original trajectories. Red and Green Dots: partners identified by my proposed algorithm.	48
5.18	Truck trajectories in Greece. Blue dots/lines: original trajectories. Red and Green Dots: partners identified by my proposed algorithm.	49
5.19	Speed-Up comparison with changing parameters and number of cores. . .	50
5.20	Graph of second time comparison, with one extra core.	50

LIST OF TABLES

	<u>Page</u>
3.1 List of symbols and notations used in Algorithms 1 and 2.	23
4.1 List of symbols and notations used in Algorithms 3 and 4.	30

CONTENTS

	<u>Page</u>
1 INTRODUCTION	1
1.1 Motivation	2
1.2 Objective	3
1.3 Contribution	4
2 THEORY AND LITERATURE REVIEW	5
2.1 Moving objects trajectory	5
2.2 Trajectory knowledge discovery and data mining	5
2.2.1 Trajectory data selection	6
2.2.2 Trajectory data preprocessing	6
2.2.3 Trajectory data reduction	7
2.2.4 Semantic enrichment of trajectories	9
2.3 Trajectory pattern identification	10
2.3.1 Moving together patterns	10
2.4 Software for moving object trajectories analysis	12
2.4.1 R	12
2.4.2 Terralib and trajectory	13
3 A FRAMEWORK FOR BIG TRAJECTORY DATA MINING	17
3.1 Framework architecture	19
3.2 TrajDataAccess package	21
3.3 TrajDataMining package	25
3.4 Comparison between proposed framework and related work	25
4 AN ALGORITHM TO DISCOVER PARTNERS IN TRAJECTORIES	27
4.1 Method and definition	27
4.2 Algorithm	29
4.3 Partner algorithm parallelization	32
4.4 Comparison among the proposed pattern and related work	34
5 CASE STUDIES	35
5.1 KML trajectory data access	35
5.2 PostGIS trajectory data access	37
5.2.1 PostGIS trajectory data access by parts	39
5.3 Trajectory manipulation and analysis	43

5.4	Partner discovery	46
5.4.1	Vessel partner	46
5.4.2	Truck partner	47
6	CONCLUSIONS AND FUTURE WORK	51
	REFERENCES	53
	APPENDIX A - LIST OF FUNCTIONS OF TRAJDATAACCESS PACKAGE	59
	APPENDIX B - LIST OF FUNCTIONS OF TRAJDATAMINING PACKAGE	61

1 INTRODUCTION

Recent advances of sensors and communication technologies have produced massive spatiotemporal data sets that allow scientists to observe the world in novel ways (FERREIRA et al., 2013). Earth observation satellites capture changes over time in cities and forests; environmental sensors measure the variation of air pollution, temperature and humidity in specific locations; and GPS (Global Positioning System) satellites and devices collect locations of animals, vehicles and people over time. Mobile phones, sensor networks, social networks and GPS devices create useful data for planning better cities, capturing human interactions and improving life quality.

Mobility of people and goods is essential in the global economy (RENZO et al., 2013). In GIScience (Geographic information science), moving object trajectories is a well-known category of spatiotemporal data, and the gathering of such data has become common in recent years (ALVARES et al., 2007). This work focuses on this category. Moving objects are entities whose spatial positions or extents change continuously over time (ERWIG et al., 1999). Examples of potentially moving objects are cars, aircraft, ships, mobile phone users, polar bears, hurricanes, forest fires, and oil spills on the sea.

Trajectories are countable journeys associated to moving objects (SPACCAPIETRA et al., 2008). It can be semantically annotated, i.e. enriched with additional information, to convey some sort of meaning about the trajectory or its segments. For example, trajectories can be segmented accordingly to specific spatial locations or speed of the moving objects, making possible to identify traffic lights or frequent stops for vehicles. Semantic trajectories are those enriched with some sort of meaning. For instance, the trajectory of an object that moves from a location A to a location B during the week, might represent anything. Knowing that A is a company and B a house, such trajectory can be enriched with meaning. In this case, such trajectory is probably a commute to work.

Recently, the research area of trajectory data mining has grown a lot and many methods for trajectory pattern discovering have been proposed. Studies on this area consist in analyzing the mobility patterns of moving objects and in identifying groups of trajectories sharing similar patterns (ZHENG, 2015). I can mention, flocks (VIEIRA et al., 2009) and convoys that look for clusters of moving objects (JEUNG et al., 2008). The avoidance pattern (LOY et al., 2010) finds areas being avoided by objects, and so on. The discovery of patterns is of great importance in a wide range of fields, for example in automated surveillance and regulation control. Fishing can be monitored to verify if it is being done according to regulation, in order to preserve marine life. Pattern

discovery can also be used for the detection of anomalies (ETIENNE, 2011), or for tourism applications, which generate revenue (RENZO et al., 2013). An example I can mention is the identification of the favorite routes taken by tourists.

1.1 Motivation

Although there are many methods proposed in the literature for trajectory data mining, there is a lack of software tools, high-level programming environments that allow users to access big trajectory data sets and easily develop new algorithms to analyze them. Two examples of software tools that handle trajectory data are M-Atlas (RENZO et al., 2013) and Weka-STPM (ALVARES et al., 2010).

M-Atlas implements a query language to work with trajectories exclusively in PostgreSQL. Weka-STPM extends Weka by implementing the finding of stops and moves in trajectories. Both M-Atlas and Weka-STPM are written in Java. They do not provide an environment with a scientific standard programming language (MÜLLNER et al., 2013) to facilitate the development of new algorithms for application users. Besides that, they are not able to access trajectories from different kinds of data sources, such as KML files, other DBMS or web services. I believe, there is a need for high-level scientific standard programming environments to handle trajectory data sets.

R is a software tool and a programming language widely used for data analysis (R Development Core Team, 2011). It provides a broad variety of statistical methods (e.g. time-series analysis, classification and clustering) and graphical techniques (e.g. plot). R provides for instance the possibility to analyze spatial and spatiotemporal data. Such variety in functions can be attributed to an ever-growing community, that frequently creates and modifies packages to extend R. Currently, there are over 8000 packages available through CRAN (The Comprehensive R Archive Network). R provides a high-level programming language and is platform independent, allowing fast developing of new algorithms with portable code. It is free and open-source. R can be bound to other languages like C++, allowing high performance computing to be executed.

Although there are many packages for spatial and spatiotemporal analysis such as *spacetime* (PEBESMA, 2012), there are few R packages that work with trajectory data. Some examples are *SimilarityMeasures* (TOOHEY; DUCKHAM, 2015), *AdehabitatLT* (CALENGE, 2006) and *Trajectories* (KLUS; PEBESMA, 2015). *SimilarityMeasures* implements trajectory similarity measures. *AdehabitatLT* analyzes trajectories of animals specifically. *Trajectories* contains classes to represent a trajectory and some operations over them, such as calculating distances and returning Spatiotemporal boundaries. These packages do not share a standard representation for trajectories, limiting

therefore the users' experience when analyzing trajectories. Besides the existing trajectory packages do not offer data access methods, they focus solely in data processing.

Trajectory data sets can be stored in different sources, such as database systems (e.g. PostGIS) and data files (e.g. KML - Keyhole Markup Language) (FERREIRA et al., 2015). In R, there are packages like RPostgreSQL (CONWAY et al., 2012) and Rgdal (BIVAND et al., 2013) that can access data from distinct sources. They can access spatial data, but they are limited when it comes to temporal dimension. Moreover, these packages are not able to access trajectory data by parts. Thus, they cannot handle big data sets.

According to Kane et al. (2013), R is not well-suited for working with data structures larger than about 10–20% of a computer RAM memory. The authors argue that a data set is considered large when its size is 20% or more of the computer RAM; and it is massive when its size is 50% or more of the computer RAM. In this work I obtained the empirical data regarding R packages in a Virtual Machine running Ubuntu 14.04.5 LTS with 3 processors and 3 GB of RAM. In this case, a 2 GB data set was already challenging.

In the R environment, there is a need for packages that can access big trajectory data sets from distinct types of sources and discover patterns, using a standardized trajectory representation.

1.2 Objective

The goal of this work is to propose a framework for big trajectory data mining that provides a high-level programming environment that allows users to efficiently deal with big trajectory data sets and quickly develop new mining algorithms over them. The framework is designed using the R environment and the GIS (Geographical Information System) library and application called TerraLib and TerraView (CÂMARA et al., 2008).

I chose R because it provides a high-level programming environment and language that is suitable for fast developing and testing of new methods. Besides that, it is extensible and provides many building blocks to boost the process of development. Visualization of spatiotemporal data is possible in R, but it is not as dynamic and effective as in a GIS. In the framework, I propose the use GIS tools to handle and visualize trajectory data sets. In this way, users can use GIS typical methods over trajectory data sets and combine such data sets with other kinds of geographical data, such as Earth Observation (EO) satellite images.

The developed framework is composed of two new R packages. The `TrajDataAccess` is responsible for accessing big trajectory data sets from distinct types of sources. The `TrajDataMining` contains a set of methods for trajectory data preparation, such as filtering, compressing and clustering, and for trajectory pattern discovery.

To validate and test the proposed framework and its packages, I performed case studies using real trajectories of marine vessels; sea elephants; delivery trucks and a trajectory provided within the package trajectories.

1.3 Contribution

The main contributions of this work are:

- A framework capable of loading and processing trajectory data made up of:
 - A new R package called `TrajDataAccess` to access big trajectory data sets from distinct types of sources;
 - A new R package called `TrajDataMining` with a set of functions to prepare trajectory data sets for further data mining processing, such as filtering, compressing and clustering;
- A new method to detect a pattern called *Partners* in trajectories. This algorithm identifies pairs of trajectories whose objects stay together during certain periods. This method is available in the `TrajDataMining` package. It differs from existing moving together patterns because its implementation is highly parallelizable and its clustering method is based on distance time series analysis.

2 THEORY AND LITERATURE REVIEW

2.1 Moving objects trajectory

Moving objects are entities in an environment whose position evolve along the time. These objects can be vehicles, human beings, animals, natural phenomena (ETIENNE, 2011). Every object has its own characteristics such as its geometry, its moving capability, and these characteristics influence their representation.

There are some ways to represent movement, for instance fields of vectors and flow network (RENZO et al., 2013). However, one of the most relevant ways nowadays is storing it as trajectories. According to Renzo et al. (2013) trajectories are segments of movement and while movement is inherently continuous, it cannot be captured as such in computers where stored data is by definition discrete.

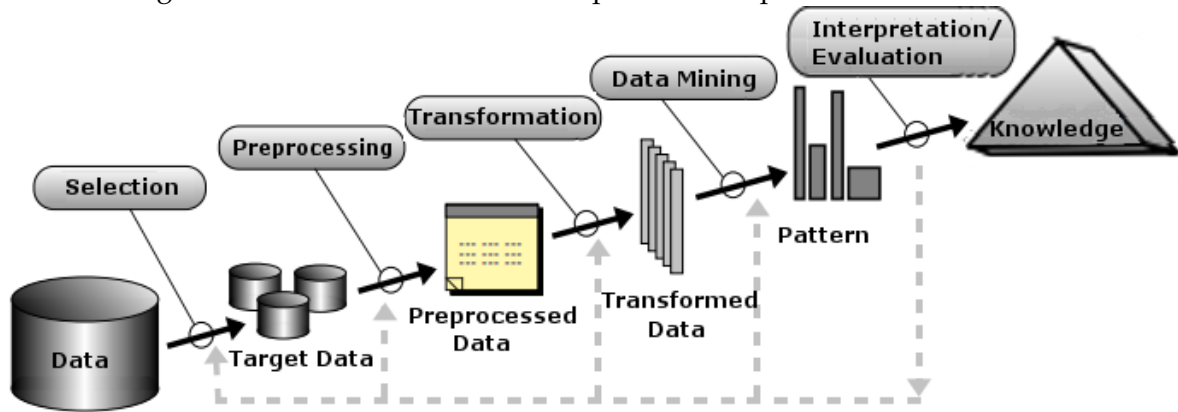
Alvares et al. (2007) states that moving object data are normally available as sample points in the form (tid, x, y, t), where tid is an object identifier and x, y and t are respectively spatial coordinates and a time stamp.

Other ways of representing trajectories exist, such as equations of motion (GEERTS, 2004), in which the trajectories per se are not stored, they are the solution to the stored equations. Nonetheless, for this work I chose to work with the discrete representation that is the most common.

2.2 Trajectory knowledge discovery and data mining

The knowledge discovery and data mining(KDD) process consists of more stages than pure data mining. The process involves the selection of the data, cleaning and pre-processing the data, followed by data reduction, data mining and analyses (FAYYAD et al., 1996). This process is illustrated in figure 2.1.

Figure 2.1 - An Overview of the Steps That Compose the KDD Process.



SOURCE: (FAYYAD et al., 1996)

2.2.1 Trajectory data selection

Data selection is important for the process of knowledge discovery, without it all the other processes could be hindered. Without the means to properly select data, worthless or excessive data might enter the process. Data selection makes possible to focus solely on regions or objects of interest.

There are many ways we could select trajectories, we could select them by their unique identifications. Trajectories could receive other information, such as the kind of moving object performing them, and have this extra information used as part of a filter.

A natural way to filter trajectories data is by using a *spatiotemporal box* (STBox). STBox is a variant of a bounding box that takes into account not only the spatial boundaries but the temporal extent as well. Using STBoxes, we can filter trajectories considering their intersections in space and time, reducing the amount of data for further processing.

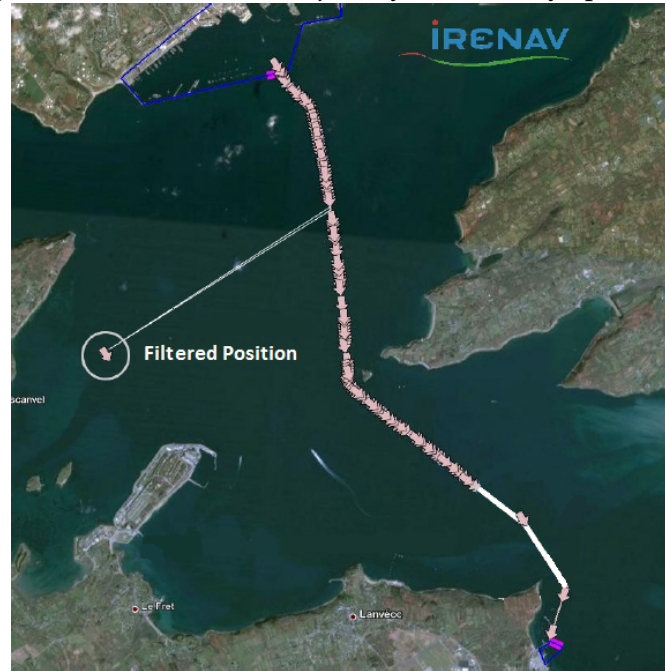
2.2.2 Trajectory data preprocessing

Data preprocessing is a step in which the data is prepared to avoid problems such as missing data, noise and inconsistencies, aiming to produce a better mining. One technique that can be used to clean trajectory data from noise is a speed filter. This filter analyses the speed presented in the data and compares it to a maximum speed (ETIENNE, 2011) and removes data in which the the speed surpasses the maximum. The maximum speed can be given by the user or calculated based on the average speed. Ideally, when setting the filter, the user is aware of the dimension of the data.

If the user sets the maximum speed too high nothing will be filtered, while if it sets too low real data might be excluded.

Figure 2.2 shows an abnormality that should be filtered. It is unlikely that a vessel could have a point so far from the rest of the path.

Figure 2.2 - Inconsistent trajectory detected by speed filter.



SOURCE: Edited from Etienne (2011)

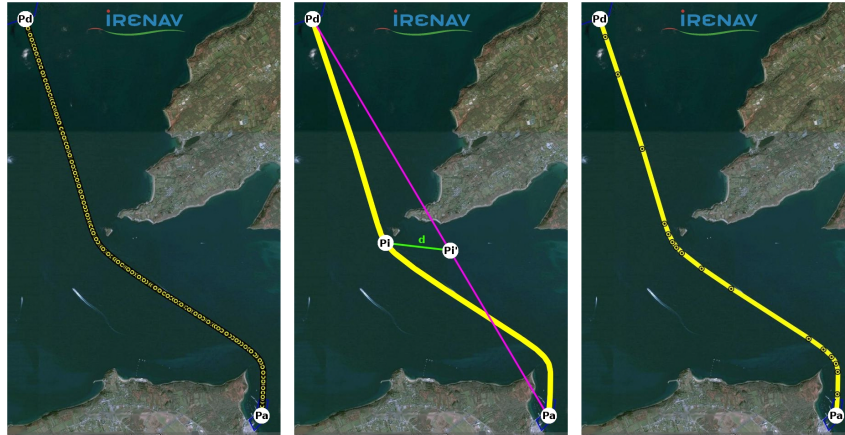
2.2.3 Trajectory data reduction

In general, data reduction can be achieved through many different techniques. Such as dimension reduction in which some variables are ignored; discretization that transform continuous data into discrete data; data compression that tries to use fewer information to represent the same data (BASKAR et al., 2013). I focus my work on the latter, since trajectory data, in this work, is already discrete and minimally represented.

When using lossy compression techniques there is always a trade-off between compression rate achieved and error allowed. It is important to notice that as compression rate increase so does the error, however at a certain point neither compression nor error will increase further (MERATNIA; BY, 2004). Because of the error produced when using lossy compression techniques not all applications can be compressed.

One well known and studied compression technique is the one proposed by [Douglas and Peucker \(1973\)](#) and that has been studied and modified along the years in works such as the one of [Vaughan et al. \(1991\)](#). This technique was designed for line simplification, and can be used to compress trajectories, even though it is not designed to keep the temporal aspects intact. This technique can be visualized in figure 2.3, in which a trajectory is reduced.

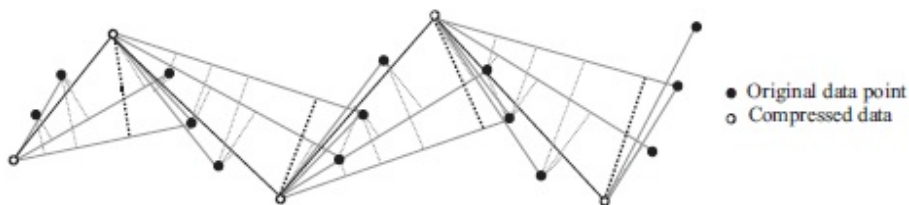
Figure 2.3 - Exemplification of Douglas-Peucker algorithm for line simplification.



SOURCE: ([ETIENNE, 2011](#))

In order to minimize the loss from [Douglas and Peucker \(1973\)](#) algorithm when dealing with spatiotemporal data, [Meratnia and By \(2004\)](#) have proposed a new technique. Their algorithm is built over an open-window compression one, exemplified in figure 2.4. It takes into consideration both speed and position when compressing data.

Figure 2.4 - Exemplification of open-window strategy.



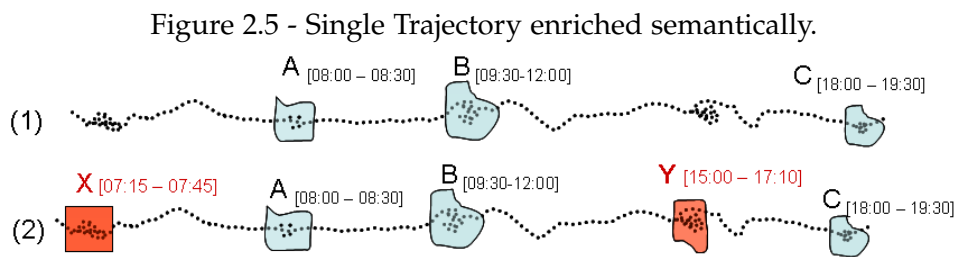
SOURCE: ([MERATNIA; BY, 2004](#))

In their work, [Meratnia and By \(2004\)](#) demonstrate the differences between DP (Douglas-Peucker) algorithm and theirs. They show that the compression rate of the DP algorithm is higher. Specially at low distance thresholds (the gap narrows as the threshold distance increases). However, the error on their algorithm, as they calculate it, is substantially lower. Thus in applications in which compression is more important than precision, DP might be more adequate. While applications in which accuracy is the most valued attribute [Meratnia and By \(2004\)](#) algorithm should be considered, if compression is necessary.

2.2.4 Semantic enrichment of trajectories

Recently the way trajectories are viewed changed and a new model, called *stops and moves*, appeared to work with them. This model is especially interesting to add semantic information to raw trajectories ([SPACCAPIETRA et al., 2008](#)). Aiming to enrich trajectories semantically with such stops and moves, two algorithms were created to identify stops and moves of trajectories without intersecting known stops. They are CB-SMoT ([PALMA et al., 2008](#)) and DB-SMoT ([ROCHA et al., 2010](#)).

The algorithm DB-SMoT takes in consideration the change of directions in order to find clusters with many direction changes. The other algorithm finds regions in which the speed is below a set threshold and defines them as clusters. Those clusters will then be considered the stops. In figure 2.5 the trajectory in item (1) is enriched only with the known stops A, B and C and in item (2) the same trajectory is also enriched with the calculated stops X and Y.



SOURCE: ([PALMA et al., 2008](#))

Having knowledge of the stop points it is possible to semantically segment a trajectory. That means we can divide the trajectories into smaller trajectories with some sort of meaning, for instance a commute to work could be separated from the rest of the trajectory.

2.3 Trajectory pattern identification

Recently, the research area on trajectory data mining has grown a lot. Studies on this area consist in analyzing the mobility patterns of moving objects and in identifying groups of trajectories sharing similar patterns. In last years, many methods and techniques for trajectory pattern discovering have been proposed to meet a broad range of applications. Zheng (2015) presents a systematic survey on the major research into trajectory data mining and classifies existing patterns in four categories: (1) *Moving together*; (2) *Clustering*; (3) *Frequent sequence*; and (4) *Periodic*. In this work, I focus on the first category.

2.3.1 Moving together patterns

Examples of patterns that discover a group of objects that move together for a certain period are *flock* (GUDMUNDSSON; KREVELD, 2006; VIEIRA et al., 2009; TANAKA et al., 2015), *group* (WANG et al., 2006), *convoy* (JEUNG et al., 2008), *swarm* (LI et al., 2010), *traveling companion* (TANG et al., 2012), *gathering* (ZHENG et al., 2013; ZHENG et al., 2014) and *co-movement* (FAN et al., 2016). *Moving together patterns* are useful for a high number of applications, such as monitoring of delivery trucks (JEUNG et al., 2008) and identification of vessels that fish together.

The *flock* pattern has attracted a lot of interest from the community with many studies being published over the years regarding this pattern. A *flock* is a group of objects that stay together within a disk with a user-defined radius for at least K consecutive time stamps. Vieira et al. (2009) propose a framework and polynomial-time algorithms, called *Basic Flock Evaluation* (BFE), to discover such pattern in streaming spatiotemporal data. Tanaka et al. (2015) present variations of the BFE algorithm, employing the plane sweeping technique, binary signatures and/or an inverted index. Similar to *flock*, *group* pattern identifies moving objects that travel within a radius for certain timestamps that are possibly nonconsecutive (WANG et al., 2006). The main difference between both is that *group* considers relaxation of the time constraint.

According to Zheng (2015), a major concern with *flock* and *group* patterns is the pre-defined circular shape, which may not well describe the shape of a group in reality. Since they use a disk with rigid limits, they miss objects that are close to a group but outside the disk limits. This drawback is called *lossy-flock* problem. The chosen disk size has a substantial effect on the results of the discovery process. The selection of a proper disk size is very difficult. Besides the *lossy-flock* problem, for some data sets, no single appropriate disc size may exist that works well for all parts of the space and time domain (JEUNG et al., 2008).

The *convoy* pattern uses density-based clustering in order to capture groups of arbitrary extents and shapes. Instead of using a rigid size disk as *flock*, such pattern requires a group of objects to be density connected during k consecutive time points. While both *flock* and *convoy* have a strict requirement on consecutive time period, Li et al. (2010) propose a more general type of trajectory pattern, called *swarm*, which captures the moving objects that move within arbitrary shape of clusters for certain timestamps that are possibly nonconsecutive. Even though *swarm* and *group* patterns consider relaxation of the time constraint, the *group* pattern definition restricts the size and shape of moving object clusters by specifying the disk radius. Moreover, redundant *group* patterns make the algorithm exponential (LI et al., 2010).

Aiming to overcome the limitations brought by the global consecutiveness of the *convoy* and aiming to be more selective than the *swarm*, Li et al. (2015) have proposed the *platoon*. The *platoon* is not as restrictive as the *convoy* regarding the consecutiveness of the timestamps, nor is as loose as the *swarm* on the same matter. It uses the concept of local consecutiveness, in which objects can separate for a while, as long as the periods in which they are together meet the minimum temporal requirements.

The *traveling companion* (TANG et al., 2012) proposes a data structure, called traveling buddy, to improve the efficiency of the algorithms to find moving together patterns from trajectories that are being streamed into a system. The concepts of *convoy* and *swarm* patterns are similar to *traveling companion*. The main difference is that *convoy* and *swarm* need to load entire trajectories into memory for a pattern mining. Hence it is impractical to use them in a data stream environment. The *traveling companion* pattern can be considered an online (and incremental) detection fashion of *convoy* and *swarm* (ZHENG, 2015). The *gathering* pattern (ZHENG et al., 2013; ZHENG et al., 2014) detects some incidents, such as celebrations and parades, in which objects join in and leave an event frequently. This pattern loses the constraints of the aforementioned patterns by allowing the membership of a group to evolve gradually (ZHENG, 2015).

Fan et al. (2016) propose a more general patterns, called *co-movement*, to unify those to identify moving together patterns, such as *convoy*, *swarm*, *flock* and *group*. They argue that *co-movement* pattern can avoid the *loose-connection anomaly* and can be reduced to any of the previous pattern by customizing its parameters. *Loose-connection anomaly* refers to the problem in which clusters that are clearly too distant in time are considered part of the same pattern. Moreover, the authors propose two types of parallel and scalable frameworks to process the *co-movement* method and deploy them on MapReduce platform.

2.4 Software for moving object trajectories analysis

One software that allows manipulating trajectories is M-Atlas ([RENZO et al., 2013](#)), which is mostly dedicated to urban mobility. It is written in Java and implements a query language to work with trajectories exclusively in PostGIS. This software presents a series of functions to deal with trajectory data and extract information and knowledge of those. Examples of functions that can be found in this software are the identification of stops in trajectories and the finding of flocks. However, the software is unable to load a vessel dataset (used in this study and which will be further explained in chapter 5) in the test machine at once. It is only able to deal with such data in parts, relying on the user to select the parts manually, and if the user is not careful the results of the newly analyzed data might replace the previous results.

Weka-STPM ([ALVARES et al., 2010](#)) is another tool to manipulate trajectory data. It extends Weka by implementing the finding of stops and moves in trajectories. The finding of stops and moves can be done through the use of some algorithms like CB-SMoT and DB-SMoT, presented in ([ALVARES et al., 2010](#)). However, this program has some limitations, like the necessity of naming the columns in a specific way, data can only be read from database systems, and there are no other methods to deal with trajectories in it, such as preparation methods and pattern identification.

Both M-Atlas and Weka-STPM are written in Java. They do not provide a high-level scientific programming language to facilitate the development of new algorithms for application users. Besides that, they are not able to access trajectories from different kinds of data sources, such as KML files, other databases or web services.

2.4.1 R

R is a software tool widely used for data analysis ([R Development Core Team, 2011](#)). It provides a broad variety of statistical methods (time-series analysis, classification and clustering) and a high-level programming environment and language, suitable for quickly developing new algorithms. However, R has a well-known limitation on handling large objects. According to [Kane et al. \(2013\)](#), R is not well-suited for working with data structures greater than about 10 - 20% of a computer RAM memory. The authors argue that a data set is considered large when its size is 20% of the RAM of the computer, and it is massive when its size is 50% of the computer RAM. There are other programming languages, such as C/C++ or Fortran, that allow quick and memory-efficient operations on massive data sets. Unfortunately, such languages are not well-suited for users who have low-level programming skills.

R is extended via packages, it has an ever-growing community, a good environment for the development of new algorithms, and can be easily used by people with little knowledge of programming (R Development Core Team, 2011). Even though it has thousands of packages to expand its capabilities, some which are directed towards spatiotemporal data mining, only few packages in R are developed to process trajectory data. Some examples are *SimilarityMeasures* (TOOHEY; DUCKHAM, 2015), *AdehabitatLT* (CALENGE, 2006) and *Trajectories* (KLUS; PEBESMA, 2015).

AdehabitatLT presents one of the first ways of representing animal trajectories, being better suited for those kinds of data. The package *SimilarityMeasures* does not have its own structure to represent trajectories, however it has methods to calculate the distance of trajectories, like Frechet Distance and DTW. One of the most important packages to mention in this topic is the *Trajectories* package. It has its own representation of trajectory data, that will be better explained in 3.1, and has some methods to work with trajectories, such as STBox identification.

These packages do not share a standard structure for representing trajectories (as can be expected from R packages). Their structures are limited by the host memory and can not support big data sets. Besides that, they focus on data processing and do not provide methods to access trajectory data sets from distinct types of sources.

Trajectory data sets can be stored in different types of sources, such as database systems (e.g. PostGIS) and data files (e.g. KML - Keyhole Markup Language) (FERREIRA et al., 2015). In R, there are packages like *RPostgreSQL* and *Rgdal* that can access data from distinct type of sources. They can access spatial data, but they are unaware of its temporal dimension. They do not work with the concept of trajectory and therefore lack when dealing with such data type. Moreover, these packages are not able to access data from sources by parts. Thus, they can not handle big data sets.

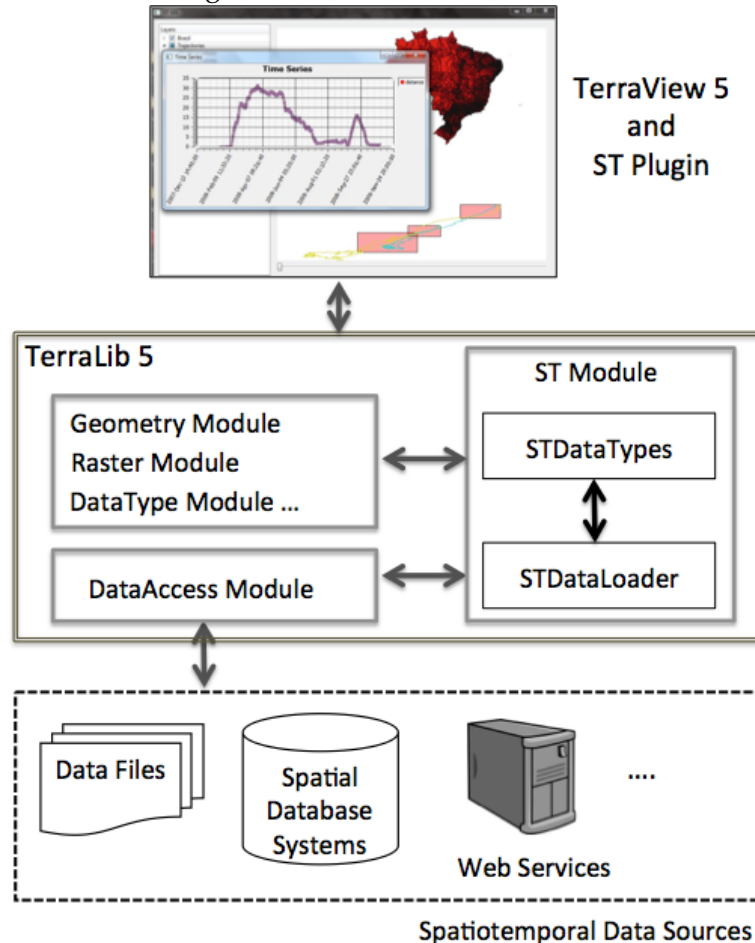
2.4.2 Terralib and trajectory

Terralib is a free and open source GIS software library to support the development of customized geographical applications. It is written in C++ language and provides typical GIS functions, such as geometry and time handling, image processing and spatial reference systems (CÂMARA et al., 2008). TerraView is a GIS built upon Terralib and can be improved via plugins.

The new family of TerraLib and TerraView versions, called TerraLib 5 and TerraView 5, can deal with spatiotemporal data (FERREIRA et al., 2015). It contains a module, called ST Module, which provides data types to represent spatiotemporal information and functions to access such information from different kinds of data sources. It is built

using other TerraLib 5 modules such as Geometry, Raster, Datatypes and DataAccess, as shown in Figure 2.6.

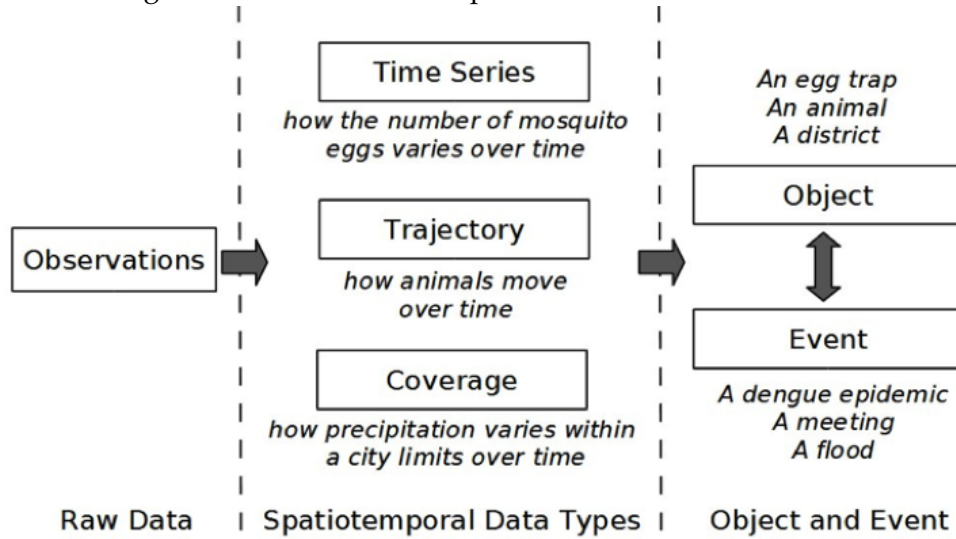
Figure 2.6 - Terralib 5 ST module



SOURCE: Author's production

In the TerraLib 5 ST Module, data types for spatiotemporal data representation were developed based on the algebra proposed by [Ferreira et al. \(2014\)](#). Algebras describe data types and their operations in a formal way, independently of programming languages. The proposed algebra is extensible, defining data types as building blocks for other types, as shown in Figure 2.7.

Figure 2.7 - Model used in part of Terralib 5 architecture.



SOURCE: (FERREIRA et al., 2014)

The proposed model takes observations as basic units for spatiotemporal data representation and allows users to create different views on the same observation set, meeting application needs. It defines three spatiotemporal data types as abstractions built on observations: time series, trajectory, and coverage, as presented in Figure 2.7. A time series represents the variation of a property over time in a fixed location. A trajectory represents how locations or boundaries of an object change over time. A coverage represents the variation of a property in a spatial extent at a time. It also defines an auxiliary type called coverage series that represents a time-ordered set of coverages that have the same boundary. Using these types, we can represent objects and fields that change over time as well as events.

The TerraLib 5 ST Module also provides a set of functions to load spatiotemporal data sets from different kinds of sources and to map such data sets into its spatiotemporal data types. Ferreira et al. (2015) present a proposal to access spatiotemporal information from distinct kinds of data sources using Semantic Web techniques. This approach consists in describing how data sources store spatiotemporal observations, based on a RDF vocabulary.

To properly visualize spatiotemporal information, a plugin for TerraView 5 will be developed. This plugin will provide graphical user interfaces (GUI) to allow users to access spatiotemporal data sets, handle these sets and dynamically visualize them in TerraView.

In 2005, [Andrade et al. \(2015\)](#) proposed and developed an interface between TerraLib and R, called aRT (R-TerraLib API) ([ANDRADE et al., 2015](#)). aRT provides access to the TerraLib entities in an easy and transparent way for R users. Even though this work is very interesting, aRT was built using TerraLib version 4 that did not have spatiotemporal data types. Therefore, it can not handle spatiotemporal information, including moving object trajectories.

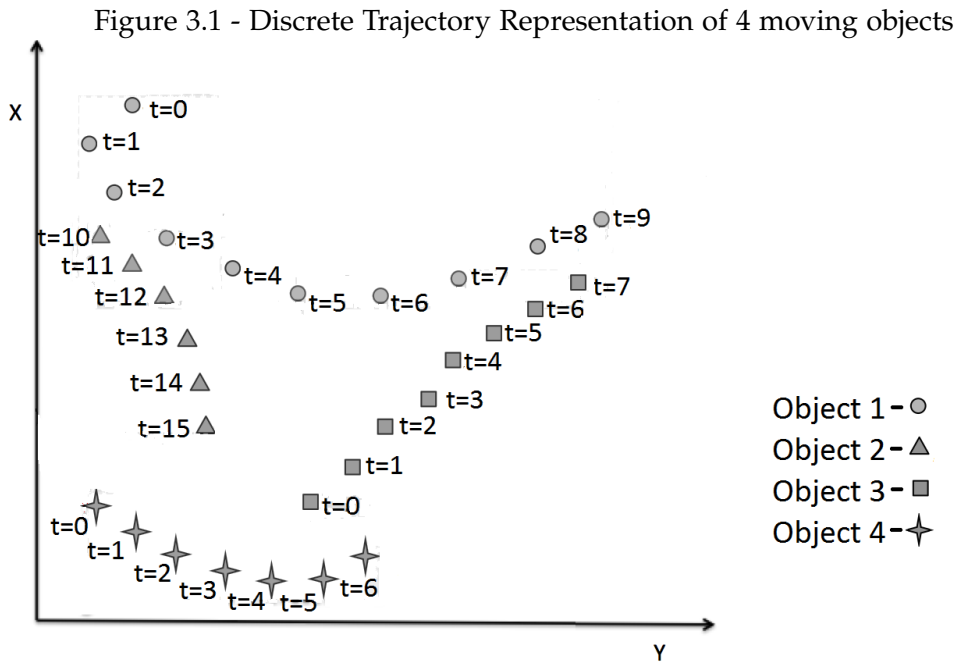
3 A FRAMEWORK FOR BIG TRAJECTORY DATA MINING

In this chapter, I describe the proposed framework for big trajectory data mining. This framework provides a high-level programming environment in R that allows users to access big trajectory data sets from different kinds of data sources and to fast develop new methods over them.

Big trajectory data mining is a relatively young topic, specially in R in which existing tools might not always be enough. So, the framework extends the R environment for users that need to handle big trajectory data sets and to fast and easily develop and test new methods on them using a high-level programming environment and language.

Before describing the framework, I present some important definitions, like the definition of trajectory and trajectory spatiotemporal box that are illustrated respectively in figures 3.1 and 3.2.

Definition 1. (Trajectory) A trajectory Trj_i , in which i is the trajectory unique identifier, is represented by a set of observations τ temporally ordered in the form $\{(x_1, y_1, t_1), \dots, (x_n, y_n, t_n)\}$. Each observation τ also represented as (x, y, t) contains the spatial location x and y of a moving object at a certain time t .



SOURCE: Author's production

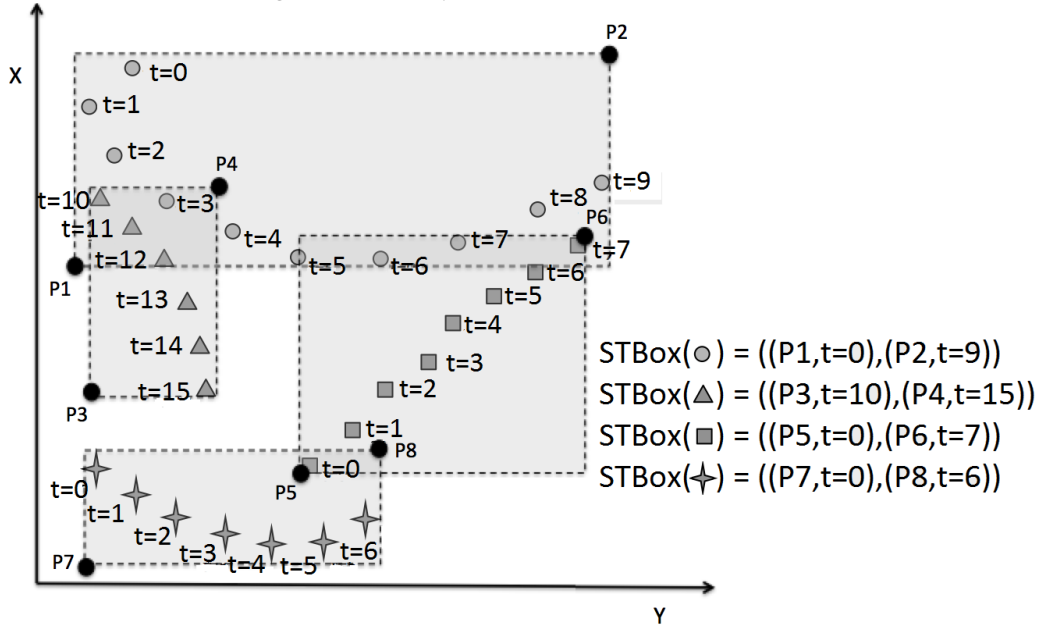
Definition 2. (Trajectory bounding box) The bounding box (BBox) of a trajectory Trj_i , $BBox(Trj_i)$, is an entity defined in space. It is defined as the rectangular region that contains all spatial locations of a trajectory. Such region is represented by two points $((x_{min}, y_{min}), (x_{max}, y_{max}))$, which are the bottom-left and top-right corners, respectively, of the rectangle given that all observations of Trj_i satisfy $x_{min} \leq x \leq x_{max}$, $y_{min} \leq y \leq y_{max}$, $\forall (x, y) \in Trj_i$.

Definition 3. (Trajectory period) The period of a trajectory Trj_i , $P(Trj_i)$, is an entity defined in time. It is delimited by (t_{min}, t_{max}) that are the minimum and maximum time instants associated to a trajectory given that all observations of Trj_i satisfy $t_{min} \leq t \leq t_{max}$, $\forall (t) \in Trj_i$.

Definition 4. (Trajectory spatiotemporal box) A spatiotemporal box (STBox) of a trajectory Trj_i , $STBox(Trj_i)$, is an entity defined in two domains, space and time. Spatially, it is defined as the $BBox(Trj_i)$. Temporally, it is given by $P(Trj_i)$. Thus, the STBox of a trajectory Trj_i can be represented as $((x_{min}, y_{min}, t_{min}), (x_{max}, y_{max}, t_{max}))$ given that all observations of Trj_i satisfy $x_{min} \leq x \leq x_{max}$, $y_{min} \leq y \leq y_{max}$ and $t_{min} \leq t \leq t_{max}$, $\forall (x, y, t) \in Trj_i$.

Definition 5. (Trajectory spatiotemporal box intersection) The intersection of $STBox(Trj_i)$ and $STBox(Trj_j)$, here represented as $STBox(Trj_i) \cap STBox(Trj_j)$, happens when $|BBox(Trj_i) \times P(Trj_i)| \cap |BBox(Trj_j) \times P(Trj_j)| \neq \emptyset$

Figure 3.2 - Trajectories and their STBoxes.



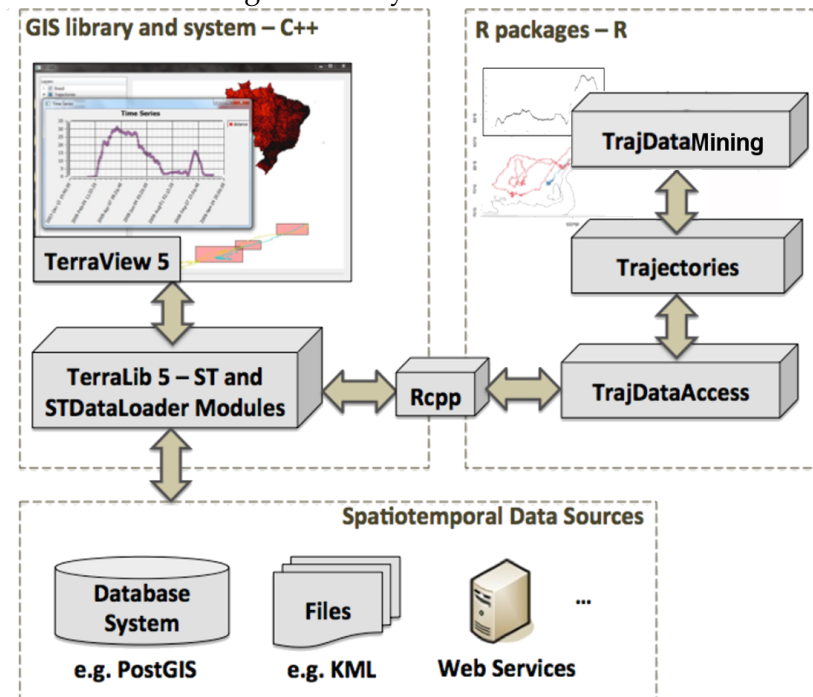
SOURCE: Author's production.

Figure 3.2 shows the trajectories of the four objects from figure 3.1 and their corresponding STBoxes. Considering the STBox of object 1, only the STBox of object 2 intersects it. STBox of object 3 intersects spatially the STBox of object 1, but not temporally. And the STBox of object 4 intersects temporally the STBox of object 1, but not spatially.

3.1 Framework architecture

I propose a framework which is composed of the GIS library and application TerraLib 5 and Terraview 5, two existing R packages, Trajectories and Rcpp, and two new R packages developed in this work called TrajDataAccess and TrajDataMining.

Figure 3.3 - System architecture.



SOURCE: Author's production.

TrajDataAccess allows R users to access big trajectory data sets from distinct types of sources and to load them as objects of the Trajectories package (KLUS; PEBESMA, 2015).

Trajectories package provides three data types to represent trajectories, Track, Tracks and TracksCollection. The class Track represents a single trajectory followed by a person, animal or object. Tracks embodies a collection of trajectories followed by a single person, animal or object. The class TracksCollection represents a collec-

tion of trajectories followed by different persons, animals or objects. Besides that, this package provides a set of operations over trajectories, such as computing the STBox of trajectories and calculating the instantaneous euclidean distance between two tracks.

An object of the `Track` class is build with observations, that have minimally a spatial position and a time of observation. A `Track` can carry other informations for every observation, such as gasoline consumption, carbon emission, but those are not mandatory. This matches the definition of trajectory presented in this work. This also matches one of the ways trajectories are represented in TerraLib 5 ST Module, in which a trajectory is made up of observations that have at least a position and the instant of the observation.

`TrajDataAccess` is responsible for accessing trajectory data from different types of sources, such as PostGIS database systems and KML files. It provides functions to load big data sets based on spatiotemporal constraints. Such functions allow users to effectively deal with big trajectory data sets by accessing them by parts. Examples of spatiotemporal constraints that can be used in `TrajDataAccess` functions are filters based on trajectories STBox, as shown in Figure 3.2. This package allows users to load trajectory data sets by parts in a high-level way, regardless of its size or how it is stored.

`TrajDataAccess` is an interface with TerraLib. In order to bind a C++ library with the R environment, I used the Rcpp middleware, which facilitates the integration between R and C++, working as a bridge (EDELBUETTEL et al., 2011). It provides matching C++ classes for a large number of basic R data types. Hence, a package author can keep his data in normal R data structures without worrying about translation or transferring to C++. At the same time, the data structures can be accessed as easily at the C++ level, and used in the normal manner.

To deal with spatiotemporal data, TerraLib has two modules called `ST` and `STDataLoader` (FERREIRA et al., 2015). The TerraLib `ST` module contains data types written in C++ classes to represent spatiotemporal data, based on the algebra proposed by Ferreira et al. (2014).

The TerraLib `STDataLoader` module is responsible for accessing different kinds of data sources, loading spatiotemporal data sets from these sources and mapping them into the data types of the `ST` Module. It accesses sources and load data using two main concepts, *Data Source* and *Data Set*. Details about TerraLib `ST` and `STDataLoader` can be found in (FERREIRA et al., 2015). Internally, the functions provided by the `TrajDataAccess` package use the ones implemented in the TerraLib `STDataLoader` module.

In the framework, I propose the use of the GIS TerraView to dynamically visualize and preprocess trajectories. To properly visualize and deal with spatiotemporal information, a plug-in for TerraView will be developed. Visualization of spatiotemporal data is possible in R, but it is not as dynamic and effective as in a GIS. Using a GIS to handle and visualize trajectory data sets, their typical methods over such data sets can be used and the data combined with other kinds of geographical data, such as Earth Observation (EO) satellite images.

The TrajDataMining contains a set of methods for trajectory data preparation, such as filtering, compressing and clustering, and for trajectory pattern discovery. The methods are important to prepare trajectory data sets before the data mining phase.

3.2 TrajDataAccess package

The TrajDataAccess R package contains functions (listed on annex A) to:

- a) Calculate the maximum data size that R can load in the user environment;
- b) Calculate the spatiotemporal bounding boxes (STBoxes) related to the maximum loadable data size;
- c) Load trajectories from distinct types of data sources, such as PostGIS database systems and KML files;
- d) Load trajectories from data sources by parts, based on a given spatiotemporal bounding box (STBox) restriction;
- e) Load trajectories from data sources by their unique identifications (ID);
- f) Load all trajectories from data source at once without restrictions when memory is not a limitation.

Aiming the good usability of the package some of these functions are hidden from the user and are only used by other functions, for instance function (a) is used by the function (b), named `getIdealSTBoxes`, or are overloaded on a single function, for example, function (f) is included in function (c). Beyond the functions, I also implemented five classes: `DataSourceInfo`, `TrajectoryDataSetInfo`, `Envelope`, `Period` and `STBox`.

The `DataSourceInfo` class represents information about a data source that contains trajectories. There are three types of data sources: files, DBMS (Database management Systems) and web services. Each type of data source is described by a specific set

of attributes. For example, to describe a DBMS data source, I have to inform its *host name*, *port number*, *database name*, a *user* and its *password*. For file-based data sources just the path where the files are stored needs to be provided.

The `TrajectoryDataSetInfo` class describes a data set that contains trajectory observations in a data source. A data source can have one or more data sets that contains trajectories. In DBMS data sources, a trajectory data set can be a table or a view; in file data sources, it can be an internal tag or a sub file. Using the `TrajectoryDataSetInfo` structure, R users inform how the trajectories are stored in a data source. For example, if the data source is a DBMS and the trajectory data set is a table of this DBMS, a user has to inform which property of this table contains the spatial locations, the times and the object and trajectory identification.

The `STBox` is related to both `Envelope`, and `Period`. `Envelope` is what I call a regular bounding box in the package, and `period` is a class containing two instants that represent beginning and end. When joining both `Envelope` and `Period` the result is an `STBox` (see definitions 2,3 and 4).

The `getIdealSTBoxes` calculates the regions for data retrieval that are neither overwhelming to the RAM memory nor so small that require excessive access to the data source. This method gets the available memory on the machine and the size of the data to be loaded. In order to assess the number of needed divisions, `getIdealSTBoxes` adopts both the previously mentioned limit of 20% of the RAM on the machine and the empirically found 4 time increase in size of the data as factors. This process to calculate the necessary divisions is shown in algorithm 1.

In line 4 of algorithm 1, the `dataSetSize` is the size of the dataset while stored in the machine before being loaded into the R environment, it is multiplied by 4 in order to represent its increase while stored in the R environment, and the `availableMem` is multiplied by 0.20 in order to respect the memory limitation imposed by R. It is important to note that this algorithm does not returns the divisions per se, it returns the number of parts in which the data should be divided.

The next step is shown in the algorithm 2, from lines 3 to 5 it starts by loading the number of divisions that will be used, checking how many rows each division will have, and retrieving the `STBox` of the whole collection of data. Next, in line 6, the algorithm is concerned about calculating the basic size of a division, it divides the dimension that will be worked by the number of necessary divisions, this aims to create at first regions that have the same size, it is important to note that the increment will always be in the measurement system used by the data. Further in lines 7 and 8,

the first attempt to create an STBox is made, in line 8 the higher bound is calculated by incrementing the lower bound with the size of a division.

From line 15 and beyond it is when the fine tuning happens. The counter on line 15 tells how many times the operation of reducing the dimension has been done, so that at each interaction the size of the worked divisions will be smaller. In line 16 the amount of rows within the calculated STBox are retrieved. And in lines 17,20 and 23 the algorithm analyze the 3 possible cases of this amount of rows. It can determine that the size is right and move to the next STBox. Alternately, it can say that the size is not adequate, if the STBox is too big it will try to reduce it's size by subtracting a bigger piece at each interaction. If it is too small, it will try to double the increment.

Summarizing, The method starts with the STBox of the lowest division, then it starts adjusting one of its dimensions (x,y or time), so it will have roughly the same amount of data as all divisions should have. The process is repeated for all divisions (always altering the same dimension). Those calculated STBoxes are returned as a list of STBox objects. After this process the user will have full access to the data, virtually effortlessly using the `getTrajectoryBySTBox` method.

Table 3.1 - List of symbols and notations used in Algorithms 1 and 2.

<code>GetComputerRAM</code>	Method that returns the total size of the RAM memory in the computer
<code>GetDataSetSize</code>	Method that returns the size in Bytes(or its multiples) of a data set given a <code>DataSetInfo</code>
<code>Ceiling</code>	Method that returns the lowest integer that is higher than the argument
<code>STB</code>	A set of STboxes
<code>GetDivisions</code>	Method that returns the adequate number of divisions for a dataset given a <code>DataSetInfo</code>
<code>GetDataSetRows</code>	Method that return the number of rows in a data set given a <code>DataSetInfo</code>
<code>GetDataSTBox</code>	Method that returns the STBox of a data set given a <code>DataSetInfo</code>
<code>GetRowsInSTBox</code>	Method that return the number of rows in a data set given a <code>DataSetInfo</code> , and the STBox of interest
$\Delta dataSTBox.dim$	The size of one dimension of the STBox named <code>dataSTBox</code> . This dimension may be either x,y or time
$newSTBox.dim.max$	Highest value of one dimension in the STBox named <code>newSTBox</code> . This dimension may be either x,y or time
$newSTBox.dim.min$	Lowest value of one dimension in the STBox named <code>newSTBox</code> . This dimension may be either x,y or time
<code>error</code>	Previously defined acceptable error in the number of rows

Algorithm 1 Find Needed Divisions

```
1: procedure GETDIVISIONS(DataSetInfo)
2:   availableMem  $\leftarrow$  GetComputerRAM()
3:   dataSetSize  $\leftarrow$  GetDataSetSize(DataSetInfo)
4:   divisions  $\leftarrow$  Ceiling( $((\text{dataSetSize} * 4) / (\text{availableMem} * 0.20))$ )
5:   Return divisions
6: end procedure
```

Algorithm 2 Find Ideal STBoxes

```
1: procedure GETIDEALSTBOXES(DataSetInfo)
2:   STB  $\leftarrow \emptyset$ 
3:   divisions  $\leftarrow$  GetDivisions(DataSetInfo)
4:   rowsPerDivision  $\leftarrow$  GetDataSetRows(DataSetInfo) / divisions
5:   dataSTBox  $\leftarrow$  GetDataSTBox(DataSetInfo)
6:   increment  $\leftarrow \Delta \text{dataSTBox.dim} / \text{divisions}$ 
7:   newSTBox  $\leftarrow$  dataSTBox
8:   newSTBox.dim.max  $\leftarrow$  newSTBox.dim.min + increment
9:   for (i in 1:divisions) do
10:    if (i  $\neq$  1) then
11:      newSTBox.dim.min  $\leftarrow$  newSTBox.dim.max
12:      newSTBox.dim.max  $\leftarrow$  newSTBox.dim.min + increment
13:    end if
14:    while (True) do
15:      counter  $\leftarrow$  1
16:      rowsInStBox = GetRowsInSTBox(DataSetInfo, newSTBox)
17:      if (rowsInStBox IN rowsPerDivision  $\pm$  error) then
18:        STB  $\leftarrow$  STB  $\cup$  newSTBox
19:        break
20:      else if (rowsInStBox > rowsPerDivision + error) then
21:        newSTBox.dim.max =  $-(\text{increment} * (1 - 0.5 / \text{counter}))$ 
22:        counter = counter + 1
23:      else if (rowsInStBox < rowsPerDivision - error) then
24:        newSTBox.dim.max = +increment
25:        counter = 1
26:      end if
27:    end while
28:  end for
29:  Return STB
30: end procedure
```

The functions of the `TrajDataAccess` package that load a trajectory returns the most adequate R objects of the `Trajectories` package. Internally, trajectories are loaded from data sources as C++ TerraLib 5 Trajectory data types, using the TerraLib functions. After that, `TrajDataAccess` package brings them to R, converting them into R objects of the `Trajectories` package. Examples of the use of these methods and classes are shown in chapter 5.

3.3 `TrajDataMining` package

The `TrajDataMining` contains a set of methods for trajectory data preparation, such as filtering, compressing and clustering, and for trajectory pattern discovery. The methods for data preparation are important to prepare trajectory data sets before the data mining phase. `TrajDataMining` contains the following methods:

- a) A speed filter that filters out trajectory observations whose speeds are above a user-defined maximum velocity (ETIENNE, 2011).
- b) Two compression algorithms: (1) Douglas-Peucker which reduces trajectories by preserving spatial precisions (DOUGLAS; PEUCKER, 1973), and (2) Open-window Meratnia-By which reduces trajectories by preserving spatiotemporal precisions (MERATNIA; BY, 2004).
- c) Two algorithms to discover when objects stop and move, CB-SMoT (PALMA et al., 2008) and DB-SMoT (ROCHA et al., 2010), which can be used to semantically enrich the trajectory data.
- d) A method, named *Partner*, that identifies objects that are moving together. I propose this method to recognize trajectories that stay together, based on trajectory distance time series analysis.

`TrajDataMining` relies on the `Trajectories` package for some of its functions, for instance the *Partner* method uses the function `Compare` from the `Trajectories` package, because this function automatically creates interpolation in the trajectories being compared, since real trajectories will rarely have their times perfectly synchronized. This allows the instantaneous distance to be calculated.

3.4 Comparison between proposed framework and related work

M-Atlas and Weka-STPM are two interesting platforms for the analysis of moving object trajectories. Nevertheless, they do not supply solutions for R users, who are an ever-growing community. Thus even though they are tools for trajectory analysis they are not in direct competition with the proposed framework.

Within R there is still much space for new implementations. One R package that stands out regarding trajectories is the `Trajectories` package. I believe that the way this package defines trajectories can be used as standard for future works. Since, their minimal representation of trajectories matches my definition 1.

The `Trajectories` package defines a class to represent trajectories, however it does not offer many methods to deal with them, this opposes my implementation that focus mainly in methods to work with the available trajectories, therefore making the packages complementary.

`AdehabitatLT` focus on the representation of animal trajectories, while my methods are more general. And `SimilarityMeasures` focus, as the name says, on similarity measures not mentioning other relevant methods such as compression of trajectories.

4 AN ALGORITHM TO DISCOVER PARTNERS IN TRAJECTORIES

In this chapter, I present a new algorithm to detect partners in trajectories and the main differences among it and existing ones for moving together patterns.

4.1 Method and definition

I define two trajectories as partners when they stay together within a maximum distance during a certain period. This pattern is useful for many applications that need identity objects that are performing activities jointly, such as marine vessels that are fishing together or trucks that are working in cooperation, as well as objects that are moving or stationary close to others.

I propose a method to recognize partners based on distance time series analyzes. For each pair of trajectories selected based on *spatiotemporal boxes*, the method calculates its distance time series, that is, a time series that represents the euclidean distance variation over time of these two trajectories. Then, the method analyzes such time series regarding user-defined restrictions that define partnership.

Definition 6.(Trajectory distance time series) The distance time series of two trajectories, symbolized here as ΔTrj_{ij} , represents the euclidean distance variation over time between the two trajectories Trj_i and Trj_j , considering their positions at the same instant. Further ΔTrj_{ij} can be described as a set of tuples in the form $\{(\Delta d_1, t_1), \dots, (\Delta d_n, t_n)\}$ in which Δd_n is the euclidean distance between the spatial locations x_n and y_n of the trajectories Trj_i and Trj_j at the instant t .

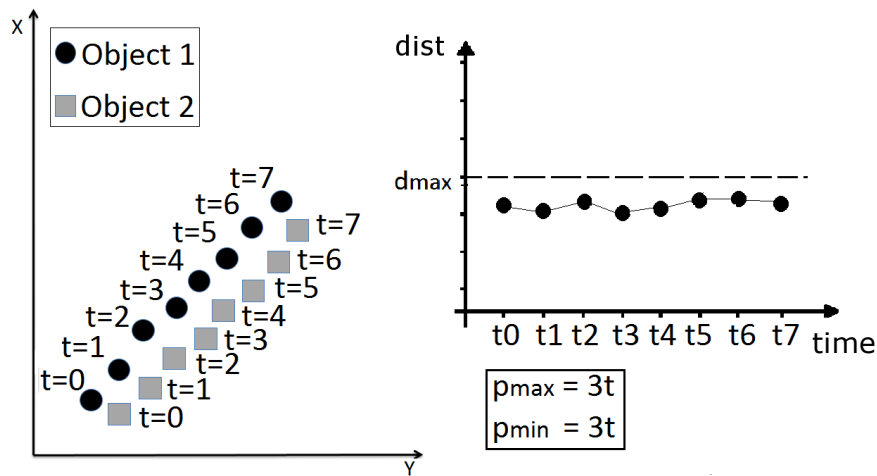
Definition 7.(Partner) Two trajectories Trj_i and Trj_j are considered partners during a certain period (t_{ini}, t_{fin}) when the distance variation between these trajectories in such period obeys two rules: (1) the distances between the two trajectories must be less than a distance threshold (d_{max}) for, at least, a minimum period (p_{min}); (2) separation periods are allowed, that is, the distance between the two trajectories can be more than a distance threshold (d_{max}) for a maximum period (p_{max}). The same pair of trajectories may be considered partner more than once on detached periods. A partner is represented by a tuple (i, j, t_{ini}, t_{fin}) , where i and j are the unique identifiers of Trj_i and Trj_j and t_{ini} and t_{fin} are respectively the initial and final instants of the period when such trajectories are partners.

After selecting trajectories based on their STBoxes, the method calculates a distance time series for each pair of trajectories and analyzes such times series regarding user-defined parameters. These parameters represent a set of restrictions that define when two trajectories are considered partners. The three parameters are: (1) a distance threshold (d_{max}), that is, a maximum distance that two objects can stay apart; (2) the minimum period (p_{min}) that two objects must stay together, that is, the distance between the two objects must be under the distance threshold (d_{max}) during, at least, the period (p_{min}); (3) a maximum period (p_{max}) that two objects are allowed to stay apart, that is, over the distance threshold. The method analyses the distance time series in order to verify whether they comply with the user-defined parameters.

Figures 4.1, 4.2, 4.3 present examples of distance time series extracted from two trajectories. Figure 4.1 shows an example of two trajectories that are partners because all their distances are less than the maximum distance threshold d_{max} . Figure 4.2 shows two trajectories that are not partners, considering the parameter values p_{max} and p_{min} showed in the picture. In this case, objects 1 and 2 go in different directions, not matching the minimum time together requirement.

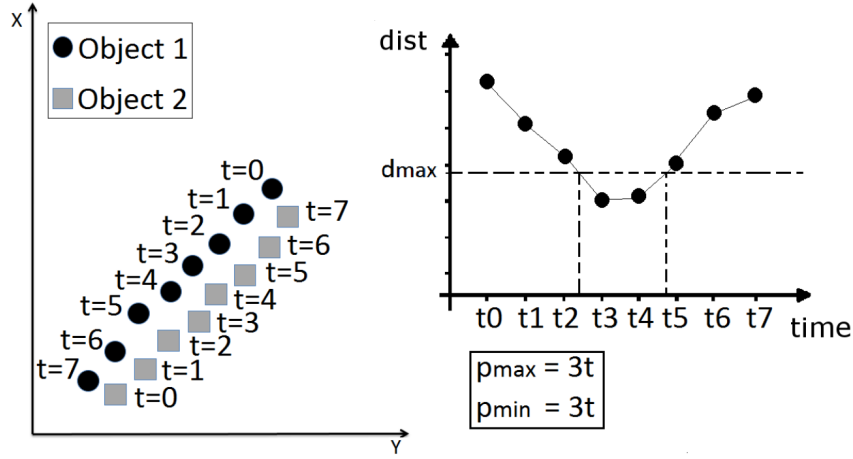
Figure 4.3 shows two trajectories that start and end together, however they stay apart for a period during their paths. Even though they surpass the maximum distance threshold d_{max} , they might be still considered partners depending on the user-defined parameter values p_{max} and p_{min} showed in the picture. Considering the parameter set (a) they are not partners. They are partners twice when the parameter set (b) is considered, and once when considering the parameter set (c).

Figure 4.1 - Example of *partner* trajectories



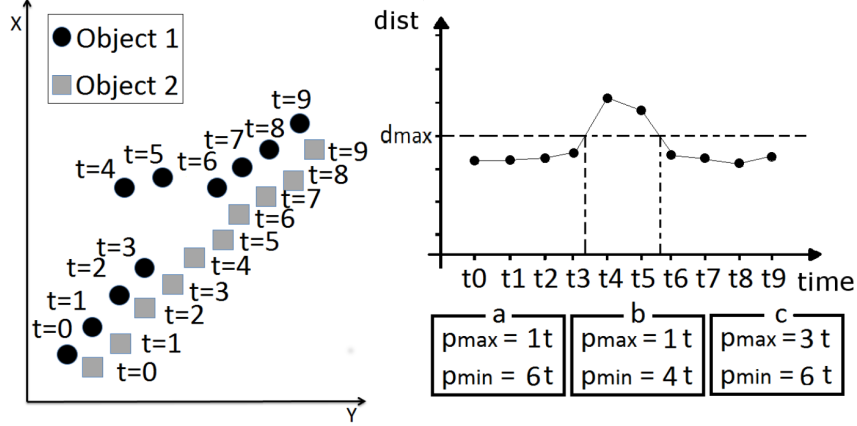
SOURCE: Author's production

Figure 4.2 - Example of trajectories which are not *partners*



SOURCE: Author's production

Figure 4.3 - Trajectories with three possible *partner* interpretations



SOURCE: Author's production

4.2 Algorithm

In order to identify partners in a dataset, I developed the two algorithms shown in Algorithm 3 and Algorithm 4. These algorithms identify possible candidates and then verify whether the candidates are indeed partners. Candidates are trajectories whose STBoxes have intersections. Symbols and notations used in both algorithms are shown in Table 4.1.

The first algorithm is responsible for the overall analysis. The user must input a set of trajectories T , and values for d_{max} , p_{max} and p_{min} . For each trajectory T_i of the set T , the algorithm repeats the following steps. The line 4 of the Algorithm 3 creates a STBox of the trajectory T_i (Definition 4) increased in all its BBox sides by d_{max} .

Table 4.1 - List of symbols and notations used in Algorithms 3 and 4.

T	A set of trajectories
$[T]$	A set of trajectories whose STBoxes have intersections
T_i	The i -th trajectory of a set (T)
d_{max}	Maximum distance that two objects can stay apart
p_{max}	Maximum time period that two objects can stay apart
p_{min}	Minimum time period that two objects must stay together
DTS	A distance time series calculated between two trajectories
DTS_i	The i -th tuple of the distance time series DTS
$DTS_i.d$	The euclidean distance component of the i -th tuple of the distance time series DTS
$DTS_i.t$	The time component of the i -th tuple of the distance time series DTS
PP	A set of time periods when two trajectories are partners
PP_i	The i -th time period of the set PP
P	A set of partners. Each partner is represented by the identifiers of the two trajectories and the period when they are close together
CreateSTBox	Method that returns the STBox of a given trajectory increased by d_{max} in both x and y directions
GetTrajectories	Method that returns the trajectories from a set whose STBoxes intersect a given STBox
CreateDistanceTimeSeries	Method that returns the distance variation between two objects over time as a time series
PeriodAsPartners	Method that returns the periods when two trajectories are partners

A STBox is created to filter only the relevant trajectories to be analyzed further. The parameter d_{max} is used to ensure that trajectories that are marginal to the STBox will be considered. This filter reduces considerably the number of trajectories being analyzed, which is important because this algorithm is computationally demanding. Its complexity is $O(n^2m)$, in which n is the number of different trajectories and m the number of observations in a trajectory. Ideally when using this algorithm, repeated verifications should be avoided, for instance, there is no need to check both T_i against T_j and T_j against T_i since the result should be the same.

Algorithm 3 *Partner Discovering*

Require: $T = \{T_0, \dots, T_n\}, d_{max} \in \mathbb{Q}, p_{max} \in \mathbb{Q}, p_{min} \in \mathbb{Q}$

```
1: procedure DISCOVERPARTNER( $T, d_{max}, p_{max}, p_{min}$ )
2:    $P \leftarrow \emptyset$ 
3:   for all  $T_i \in T$  do
4:      $sb \leftarrow \text{CreateSTBox}(T_i, d_{max})$ 
5:      $[T] \leftarrow \text{GetTrajectories}(sb, T)$ 
6:     for all  $T_j \in [T]$  parallel do
7:        $DTS \leftarrow \text{CreateDistanceTimeSeries}(T_i, T_j)$ 
8:        $PP \leftarrow \text{PeriodAsPartners}(DTS, d_{max}, p_{max}, p_{min})$ 
9:       if  $PP \neq \emptyset$  then
10:         $\forall PP_i \in PP : P \leftarrow P \cup (PP_i, i, j)$ 
11:       end if
12:     end for
13:   end for
14:   Return  $P$ 
15: end procedure
```

After creating a STBox, line 5 of Algorithm 3 retrieves the trajectories from the set T which intersect the STBox. Trajectories in T have their distance time series calculated (Definition 3) and such time series are analyzed independently by the partners verification algorithm (shown in Algorithm 4). Trajectories that are partners, according to Definition 4, are then saved in the set P .

Algorithm 4 is responsible for the fine-tuning of the partner discovery. This algorithm receives the distance time series DTS calculated between the two trajectories T_i and T_j . It checks all tuples of DTS (Definition 3) and verifies if there are one or more periods when the trajectories T_i and T_j can be considered partners, taking into account the user-defined requirements through the input values d_{max} , p_{max} and p_{min} .

Algorithm 4 checks if the distance time series DTS verifies the two rules described in Definition 4. Finally, it returns a list with all periods when the trajectories T_i and T_j are partners.

The proposed algorithms identify pairs of trajectories that are partners, returning the periods when they are nearby. Based on these pairs and periods, groups can be inferred of trajectories that are moving together. For instance, if the pair of trajectories T_i and T_j are partners from t_i to t_{i+10} and the trajectories T_i and T_k are partners from t_{i+5} to t_{i+15} , then the trio is a group from t_{i+5} to t_{i+10} .

Algorithm 4 *Partner Verification*

Require: $DTS = \{DTS_1, \dots, DTS_n\}$, $d_{max} \in \mathbb{Q}$, $p_{max} \in \mathbb{Q}$, $p_{min} \in \mathbb{Q}$

```
1: procedure PERIODASPARTNERS( $DTS, d_{max}, p_{max}, p_{min}$ )
2:    $PP \leftarrow \emptyset$ 
3:    $begin \leftarrow NULL$ 
4:    $end \leftarrow NULL$ 
5:    $timeAway \leftarrow NULL$ 
6:   for all point  $DTS_i \in DTS$ , from 1:n do
7:     switch  $DTS_i$  do
8:       case  $DTS_i.d \leq d_{max}$  and  $begin = NULL$ 
9:          $begin \leftarrow DTS_i.t$ 
10:      case  $DTS_i.d \leq d_{max}$  and  $timeAway \neq NULL$ 
11:         $timeAway \leftarrow NULL$ 
12:      case  $DTS_i.d \leq d_{max}$  and  $begin \neq NULL$  and  $DTS_i = LAST$ 
13:        if  $DTS_i.t - begin > p_{min}$  then
14:           $end \leftarrow DTS_i.t$ 
15:           $PP \leftarrow PP \cup \{(begin, end)\}$ 
16:        end if
17:      case  $DTS_i.d > d_{max}$  and  $begin \neq NULL$  and  $timeAway = NULL$ 
18:         $timeAway \leftarrow DTS_i.t$ 
19:      case  $DTS_i.d > d_{max}$  and  $begin \neq NULL$  and  $DTS_i.t - timeAway > p_{max}$ 
20:        if  $timeAway - begin > p_{min}$  then
21:           $end \leftarrow timeAway$ 
22:           $PP \leftarrow PP \cup \{(begin, end)\}$ 
23:           $end \leftarrow begin \leftarrow NULL$ 
24:        end if
25:      case  $DTS_i.d > d_{max}$  and  $begin \neq NULL$  and  $DTS_i = LAST$ 
26:        if  $timeAway - begin > p_{min}$  then
27:           $end \leftarrow timeAway$ 
28:           $PP \leftarrow PP \cup \{(begin, end)\}$ 
29:        end if
30:   end for
31:   Return  $PP$ 
32: end procedure
```

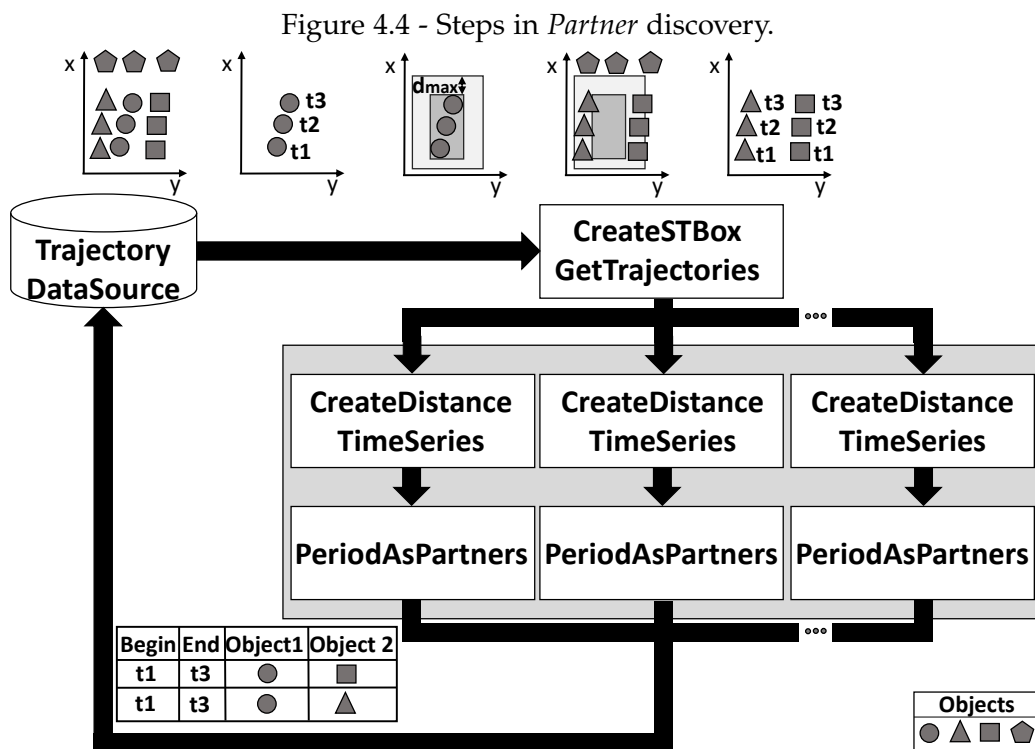
4.3 Partner algorithm parallelization

Parallel computing is one of the major techniques of High-Performance Computing (HPC). The main idea of HPC is to solve large problems faster than it would be possible using standard methods (KUMAR et al., 2003), and is a very desirable feature of systems that must process high volumes of data.

Among the many parallel computing techniques, Fork/Join parallelism is one of the most well-know, easily implemented and effective design techniques of parallel computing. Fork/Join algorithms are parallel versions of familiar divide-and-conquer algorithms (LEA, 2000).

An important feature of the proposed method is the capability of the discovery process being executed in parallel. Given the independent nature of time series analysis and partners verification, Fork/Join algorithm can be used to increase processing speed, since evaluation of one pair of trajectories does not have any influence on the others.

Figure 4.4 shows how part of the partners discovery algorithm can be executed in parallel. In this example, the distance time series for the trajectories are created and analyzed separately. The analysis results are joined by saving the set of partners.



SOURCE: Author's production

Another possible way to parallelize this algorithm is to do the task division on the first loop of the algorithm (line 3 of Algorithm 3). In this case, the processes of selecting trajectories, creating their STBoxes (function **CreateSTBox**) and filtering the trajectories whose STBoxes have intersections (function **GetTrajectories**) are executed in parallel.

Even though both parallelization alternatives are possible and can generally improve the time of the algorithm, both have shortcomings. The adequate placement of the parallel loop should be chosen according to the user needs.

When the parallelization is done on the second loop (line 6 of Algorithm 3), it might bring no improvements in areas of low trajectory density. Such lack of improvement occurs because in low density areas the trajectories generally do not have many possible partners. Therefore, in these cases, divide and conquer techniques are irrelevant. On the other hand, if the parallelization is done on the first loop (line 3 of Algorithm 3), it might consume too much memory, specially in high density areas where there are many possible partners.

It is important to mention that the parallel version of the proposed partner method may be useful only for large amounts of data. For smaller datasets the time to divide and parallelize the tasks can make the total processing time larger and not smaller.

4.4 Comparison among the proposed pattern and related work

All methods to identify moving together objects presented in section 2.3.1 are based on two steps: (1) cluster the objects of each snapshot and (2) intersect the clustering results to retrieve moving-together objects. Both clustering and intersection steps involve high computational overhead. Differently from these methods, I propose a new approach, called *Partner*, to detect moving together objects, based on trajectory distance time series analysis. The method *Partner* does not use disk with predefined circular shape, or density-based clustering, or cluster of objects. The method *Partner* analyzes the distance time series of each pair of trajectories whose spatiotemporal boxes intersect.

The main advantages of my approach are: (1) the distance time series analyses are completely independent and so can be processed in parallel; (2) users do not have to predefine either the shape and density of a group or the number of objects in a group. From the resulting pairs of partners, users can easily extract all trajectories that stayed together; (3) the method *Partner* finds objects that stayed together for a certain period, including objects that were moving as well as stationary; (4) Partners are allowed to stay apart for a maximum user-defined time; (5) the method *Partner* does not suffer the loose-connection problem, since it verifies if two trajectories are not too far for too long; (6) the method *Partner* does not suffer the lossy-flock problem, since it is not based on a rigid circular disk; (7) As long as the time away period is adequate it is noise resistant.

5 CASE STUDIES

To test and validate the implemented algorithms and framework, I have used four data sets: (1) Vessels in PostGIS, (2) Sea Elephants in KML, (3) a Track distributed with the trajectories package and (4) a group of concrete delivery trucks from Athens. I chose different datasets so that different functionalities would be well represented. This does not mean that each functionality only works with a specific dataset, it only means that the visualization is improved with different sets.

5.1 KML trajectory data access

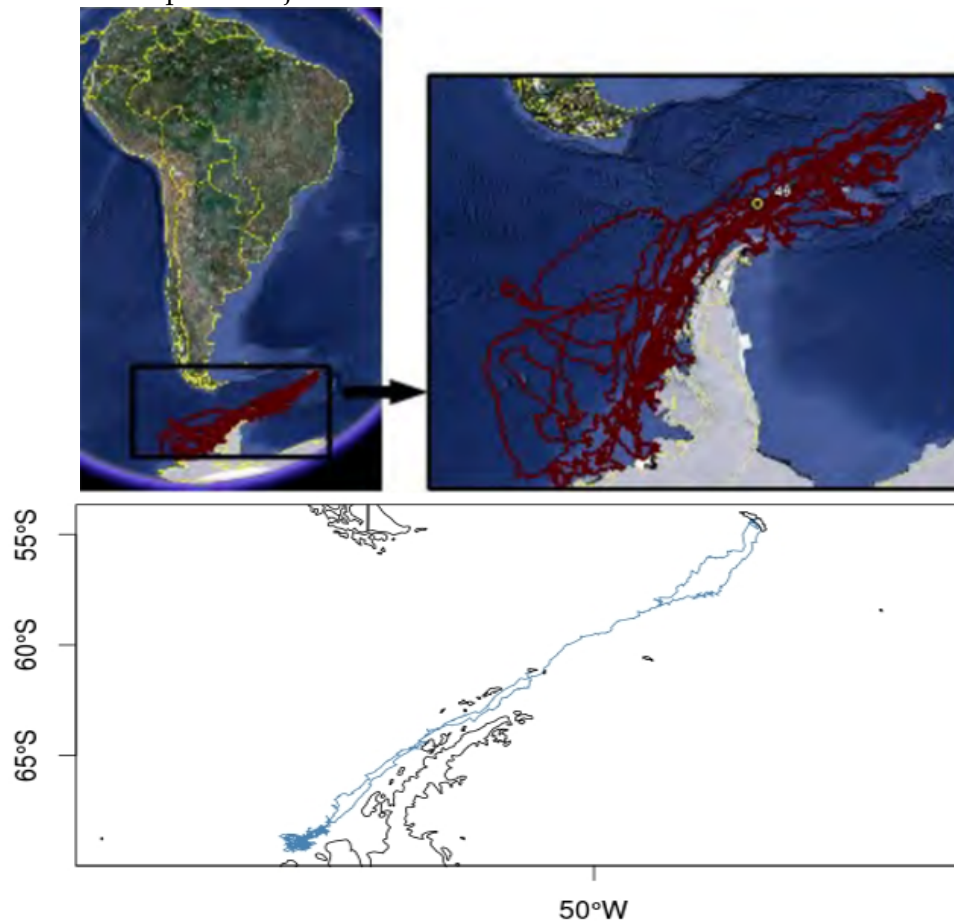
I present a case study using a KML file that contains trajectories of eight sea elephants in Antarctica during 3 years. These animals were monitored by a project called MEOP - “Marine Mammal Exploring the Oceans Pole to Pole” (<http://www.inpe.br/crs/pan/pesquisas/telemetria.php>). Figure 5.1 (above) shows all sea elephant trajectories from the KML displayed in Google Earth as red lines.

Figure 5.2 presents the R script using the `TrajDataAccess` package to access the sea elephant trajectories from the KML file. In this script, the user sets the path and the name of the KML file through the structure `DataSourceInfo`. Then, he/she informs how the trajectories are stored in this file, such as which properties contain geometries and times, through the structure `TrajectoryDataSetInfo`. In this example, I am loading the trajectories of two animals whose ids are 40 and 41, as informed in the parameter `objId`.

The function `getTrajectory` is responsible for accessing these two trajectories from the KML file and transforming them into objects of the `Trajectories` package. The objects `seaElephant 40` and `seaElephant 41` are instances of the `Track` class. Figure 5.1 (below) shows the trajectory of the animal 40 (R plot of the object `seaElephant 40`) and Figure 5.3 (above) presents the trajectories of the two animals 40 and 41 (R plot of the objects `seaElephant 40` and `seaElephant 41`).

In the last line of the code shown in Figure 5.2, I use the function `compare` of the `Trajectories` package. This function calculates the variation of the distance between two trajectories over time and returns a time series. Figure 5.3 shows the result of this code line. This figure shows a time series that represents the distance variation between the trajectories of the two sea elephants 40 and 41.

Figure 5.1 - (Above) R Plot of trajectories of sea elephant 40 and (below) Google Earth plot of all sea elephant trajectories.



SOURCE: Author's production

Figure 5.2 - R script using TrajDataAccess package to select trajectories from a KML file.

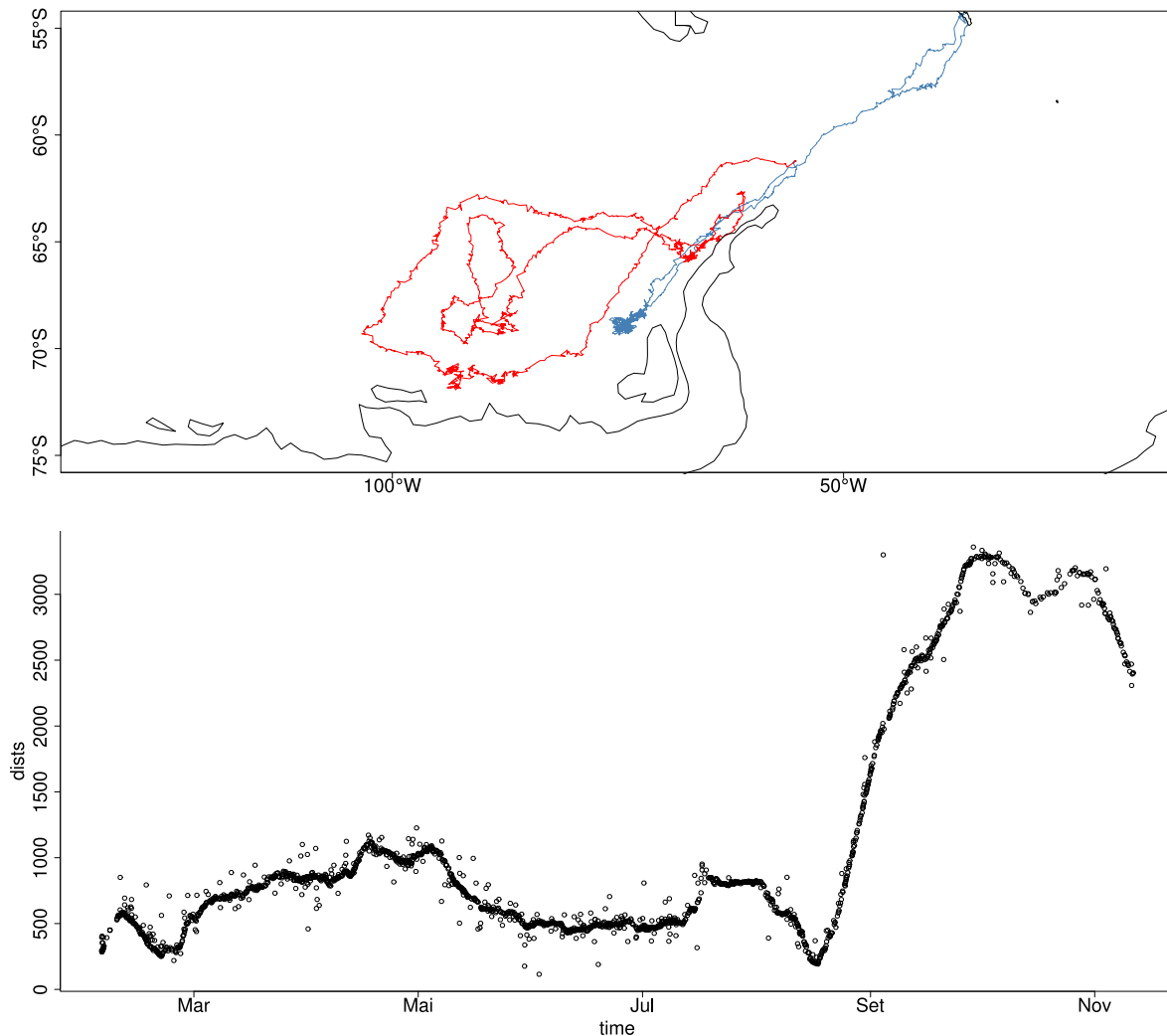
```
dsoik<-DataSourceInfo(path="/home/user/Documents/data/kml/t_40_41.kml")
dsetk<-TrajectoryDataSetInfo(dataSetName="40: locations",phTimeName="timestamp",
                             geomName="OGR_GEOMETRY",trajName="",objId="40",trajId="")
seaElephant40<-getTrajectory(dsoik,dsetk)

dsetk<-TrajectoryDataSetInfo(dataSetName="41: locations",phTimeName="timestamp",
                             geomName="OGR_GEOMETRY",trajName="",objId="41",trajId="")
seaElephant41<-getTrajectory(dsoik,dsetk)

timeseries<-compare(seaElephant40,seaElephant41)
```

SOURCE: Author's production

Figure 5.3 - (Above) R Plot of trajectories of sea elephants 40 and 41 (below) R Plot of their distance time-series.



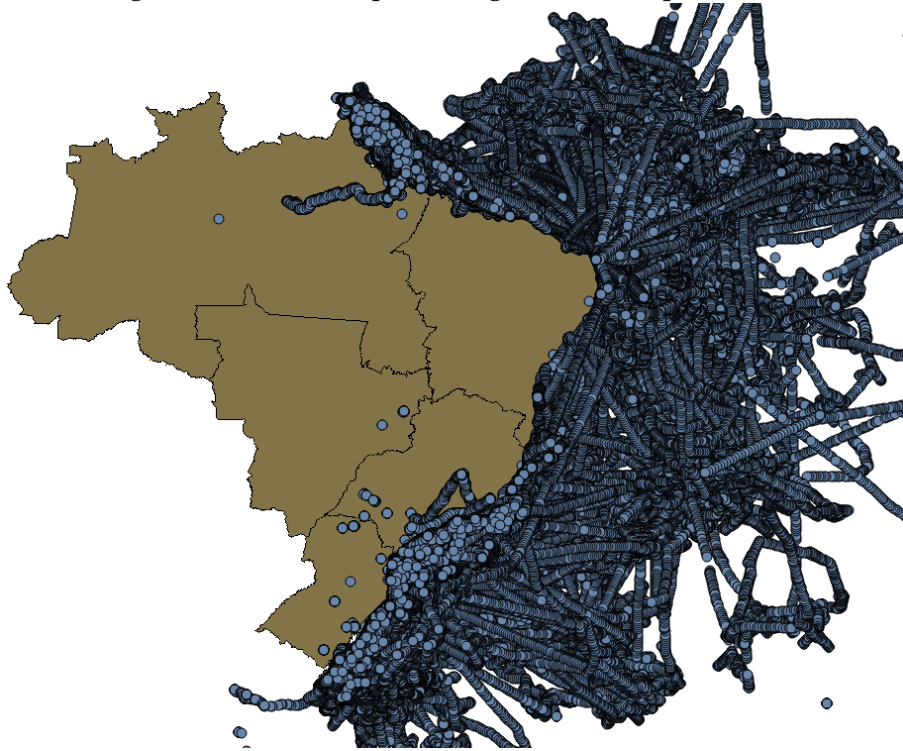
SOURCE: Author's production

5.2 PostGIS trajectory data access

In this case study, I use trajectories of 993 vessels around the Brazilian coast collected during 3 years, from 2008 to 2011. These trajectories are stored in a PostGIS database that has over 2GB of data, in more than 22 million rows. Figure 5.4 presents the trajectories of all vessels displayed in Terraview GIS.

All trajectories of all vessels are stored in a PostGIS table whose structure is shown in Figure 5.5. Each row of this table contains an observation of a trajectory of a vessel. Each observation contains a trajectory id (integer type), a vessel id (integer type), a time (timestamp type) and a spatial location (geometry type). A vessel contains one

Figure 5.4 - All data plotted against the map of Brazil.



SOURCE: Author's production

or more trajectories associated to it. Every time the vessels leave the Brazilian coast line, they start a new trajectory. And, every time they re-enter the coast, they end a trajectory. Thus, a trajectory of a vessel consists in a path traveled during the period when the vessel left the coast and re-entered the coast. The database contains 123,240 trajectories.

Figure 5.5 - Two trajectories from the same vessel.

	traj_id integer	vessel_id integer	datahora timestamp without time zone	ponto geometry(Geometry,4326)	p_key bigint	traj_id_unique bigint
1	1	2054	2008-11-26 21:14:46	0101000020E61000005D6DC5FEB22	1	20541
2	1	2054	2008-11-26 22:13:48	0101000020E6100000FD87F4DBD73	2	20541
3	1	2054	2008-11-26 23:13:46	0101000020E61000003411363CBD3	3	20541
4	1	2054	2008-11-27 00:14:18	0101000020E6100000CE88D2DEE03	4	20541
5	1	2054	2008-11-27 01:13:21	0101000020E610000040A4DFBE0E2	5	20541
6	2	2054	2008-11-27 17:15:16	0101000020E6100000A2B437F8C21	6	20542

SOURCE: Author's production

The existing R packages for loading data from POSTGIS do not work with the concept of trajectory and so are not able to access this database properly. In this case, I need to load this database by parts because of its size and the R memory limitations. Thus,

I propose the framework presented in this dissertation. I implemented the package `TrajDataAccess` that can access big trajectory data by parts, based on spatiotemporal constraints. Figure 5.6 presents a R script using the `TrajDataAccess` functions to select only the vessel trajectories in the Rio de Janeiro state, using a filter based on a `STBox`, as illustrated in Figure 3.2.

Figure 5.6 - R script using `TrajDataAccess` package to select trajectories from a PostGIS database based on a spatiotemporal constraint.

```
dataSourceInfo<-DataSourceInfo(user="postgres",password="password",db="vessel_trajectory")
trajectoryDataSetInfo<- TrajectoryDataSetInfo(dataSetName="vessel_trajectories",
                                              phTimeName="datahora",geomName="ponto",
                                              trajId="traj_id_unique",
                                              trajName="",objId="vessel_id")

envelope<-Envelope(xMax=-11.38,xMin=-44.38,yMax=-21.81,yMin=-25.03)
period<-Period(tMin="2008-01-21 06:10:37", tMax="2010-10-06 06:46:25")

accessedTracks<- getTrajectoryBySTBox(dataSourceInfo,trajectoryDataSetInfo,envelope,period)
```

SOURCE: Author's production

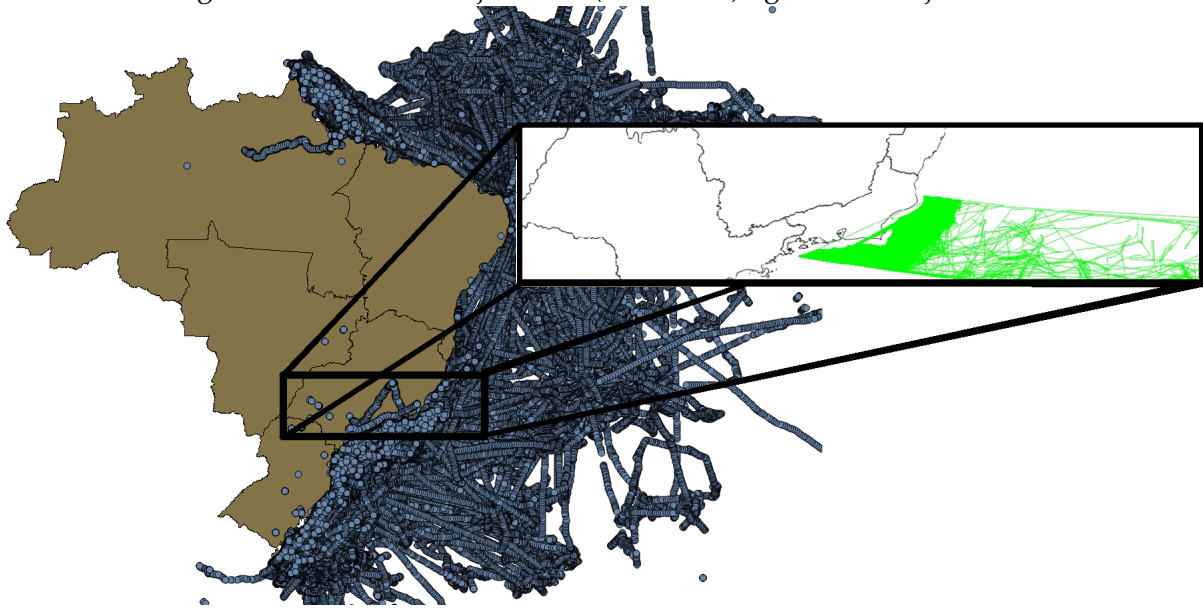
In the code shown in Figure 5.6, a user sets the necessary parameters to connect to a PostGIS database, through the structure `DataSourceInfo`. Information about the database table that stores trajectories is indicated through the structure `TrajectoryDataSetInfo`. Then, a user utilizes the function `getTrajectoryBySTBox` to load all vessel trajectories that intersect the spatiotemporal box defined by the structures `Envelope` and `Period`. Figure 5.7 shows the trajectories selected by this script in the R environment.

The function `getTrajectoryBySTBox` is responsible for loading all trajectories from the PostGIS database whose spatiotemporal boxes intersect the given envelope and period as well as for transforming them into objects of the package `Trajectories`. In this example, the object `accessedTracks` is an instance of the `TracksCollection` class, containing trajectories of distinct vessels.

5.2.1 PostGIS trajectory data access by parts

Even though in the first case study I was already able to select the data in parts with no technical problems. The question of how should the data be divided still exists. Therefore, I continue the work with the vessel trajectory data set, aiming to create adequate divisions. To create the list of adequate `STBoxes`, I input the `DataSourceInfo` and `TrajectoryDataSetInfo`.

Figure 5.7 - Selected trajectories (in the box) against all trajectories



SOURCE: Author's production

Figure 5.8 - Code to get STBoxes of adequate size

```
dataSourceInfo<-DataSourceInfo(user="postgres",password="teste2ou3",db="vessel_trajectory")
trajectoryDataSetInfo<- TrajectoryDataSetInfo(dataSetName="vessel_trajectories",
                                              phTimeName="datahora",geomName="ponto",
                                              trajId="traj_id_unique",
                                              trajName="",objId="vessel_id")

idealStBoxes <- getIdealSTBoxes(dataSourceInfo,trajectoryDataSetInfo)

Tracks<- getTrajectoryBySTBox(dataSourceInfo,trajectoryDataSetInfo,idealStBoxes[[1]])
```

SOURCE: Author's production

Then I get as a result the list presented on figure 5.9 of 13 STBoxes that have approximately 2 million rows, most of those STBoxes can be visualized in figure 5.10. To get the trajectories within any of those STBoxes the user can use a variation of the `getTrajectoryBySTBox` method, in which the user inputs an STBox object instead of a Period and an Envelope. Using this method leads to figure 5.11 in which the plot of the Tracks from the first STBox from figure 5.9 can be seen.

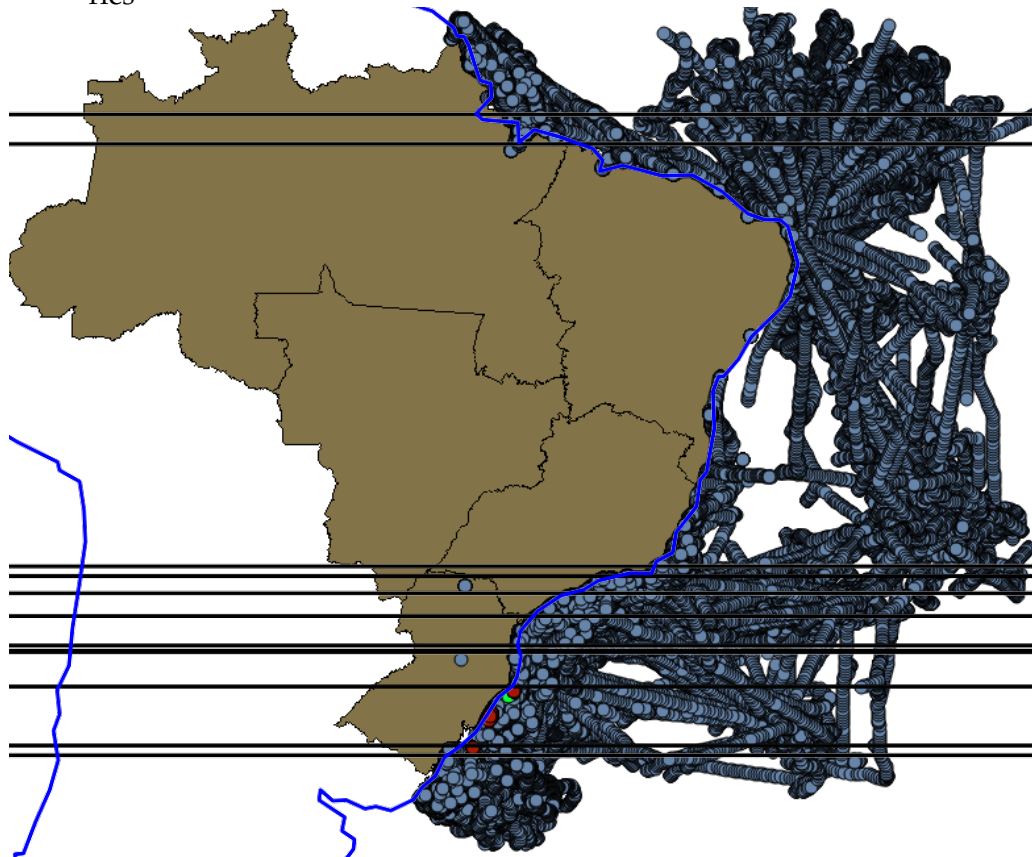
Figure 5.9 - Beginning of the list of obtained STBoxes

idealStBoxes	List of 13
:Formal class 'STBox' [package "TrajDataAccess"] with 8 slots	
.. ..@ tMin : chr "2008-01-01 00:00:21"	
.. ..@ tMax : chr "2010-10-06 07:00:20"	
.. ..@ tZone: chr "BRST"	
.. ..@ xMax : num 50.8	
.. ..@ yMax : num -32.2	
.. ..@ xMin : num -155	
.. ..@ yMin : num -69	
.. ..@ srid : chr "4326"	
:Formal class 'STBox' [package "TrajDataAccess"] with 8 slots	
.. ..@ tMin : chr "2008-01-01 00:00:21"	
.. ..@ tMax : chr "2010-10-06 07:00:20"	
.. ..@ tZone: chr "BRST"	
.. ..@ xMax : num 50.8	
.. ..@ yMax : num -31.7	
.. ..@ xMin : num -155	
.. ..@ yMin : num -32.2	
.. ..@ srid : chr "4326"	
:Formal class 'STBox' [package "TrajDataAccess"] with 8 slots	
.. ..@ tMin : chr "2008-01-01 00:00:21"	
.. ..@ tMax : chr "2010-10-06 07:00:20"	
.. ..@ tZone: chr "BRST"	
.. ..@ xMax : num 50.8	
.. ..@ yMax : num -28.7	
.. ..@ xMin : num -155	
.. ..@ yMin : num -31.7	
.. ..@ srid : chr "4326"	

In each element of the idealStBox list tMin and tMax represent the temporal constraints within the time zone tZone. xMin and xMax represent the spatial constraint in the x axis. yMin and yMax represent the spatial constraint in the y axis. xMin, xMax, yMin and yMax all use the SRID srid.

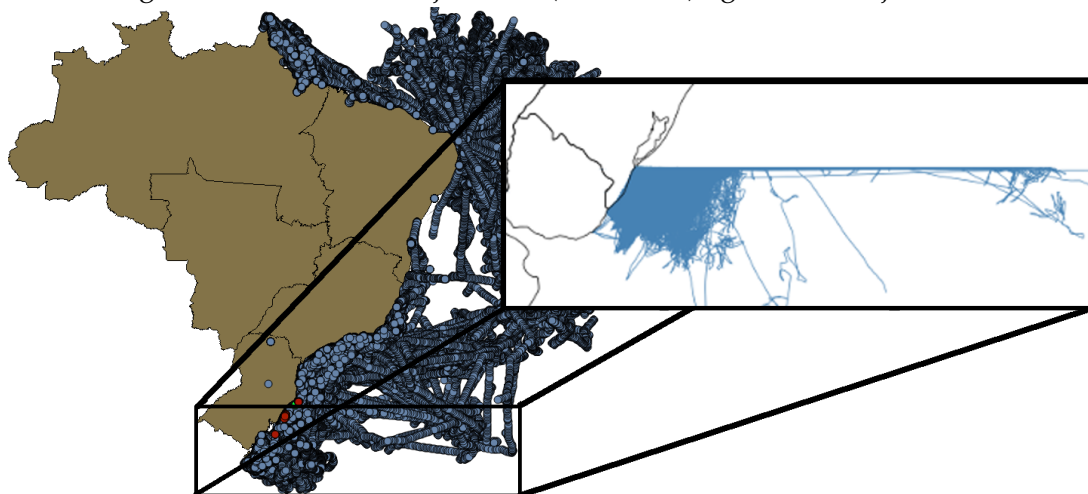
SOURCE: Author's production

Figure 5.10 - Some parts of the obtained STBoxes (horizontal black lines) against the trajectories



SOURCE: Author's production

Figure 5.11 - Selected trajectories (in the box) against all trajectories

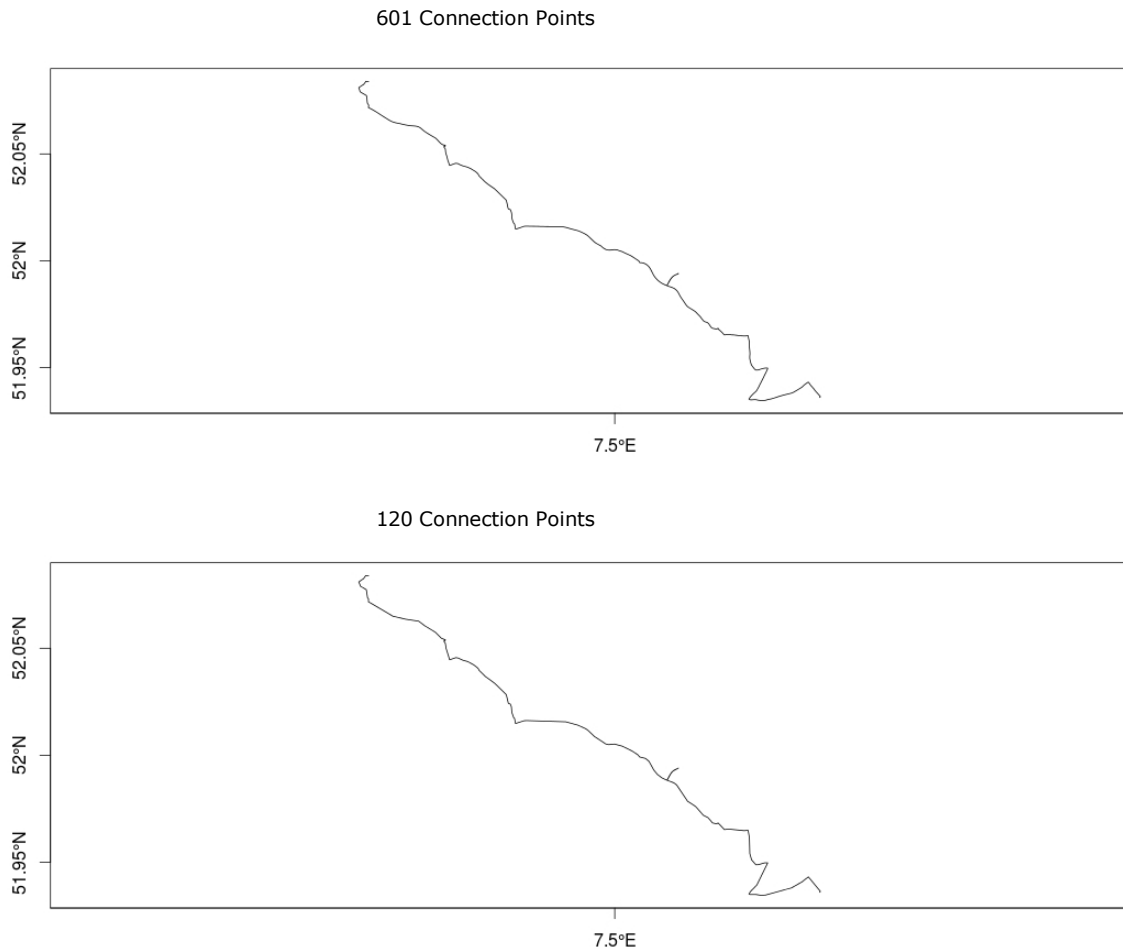


SOURCE: Author's production

5.3 Trajectory manipulation and analysis

In order to demonstrate the trajectory manipulation methods, I selected a trajectory that comes within the package Trajectories. This trajectory was selected because it contains a few curves, which allows for better visualization of the results.

Figure 5.12 - Original trajectory (above) and Douglas-Peucker compression (below), no visible changes.



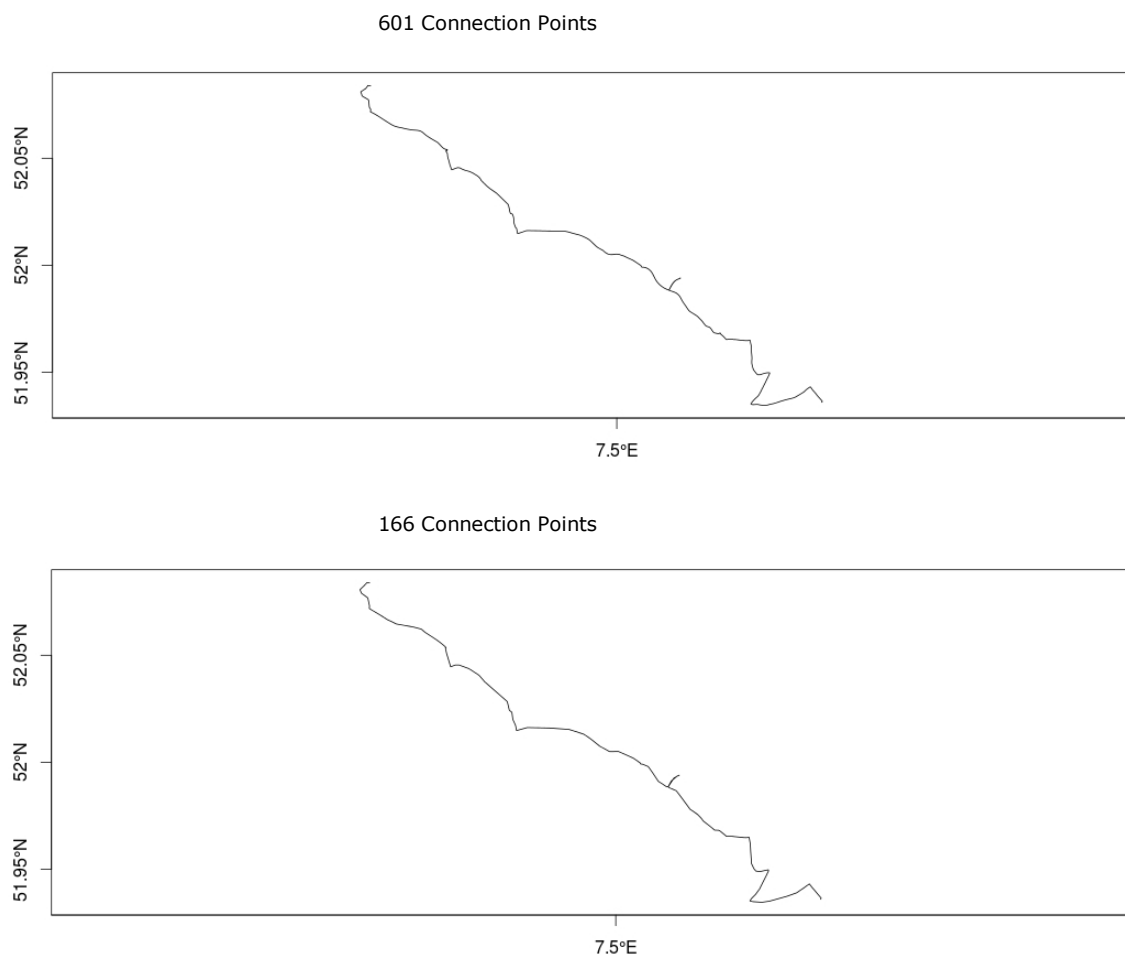
SOURCE: Author's production.

I executed the Douglas-Peucker algorithm with a tolerance of 10 meters and at another moment the open-window Meratnia-By with a threshold of 10 meters and 1 m/s. As can be seen in Figures 5.12 and 5.13 the geometry of the trajectory is maintained in both algorithms.

Both algorithms presented an approximate reduction of at least 80% in the number of points. It is important to notice that the open-window Meratnia-By algorithm preserves not only the form of the trajectories but also the speeds of the vessels. This results are similar to the ones obtained by [Meratnia and By \(2004\)](#).

I also compressed the trajectories selected by the STBox in the region of Rio de Janeiro. This dataset originally occupied 7.5 MB on R and after the compression it was reduced to 6.5 MB. This compression was not so efficient as the ones demonstrated by the figures 5.12 and 5.13 because many of the selected trajectories were small. They only had between 2 and 10 points.

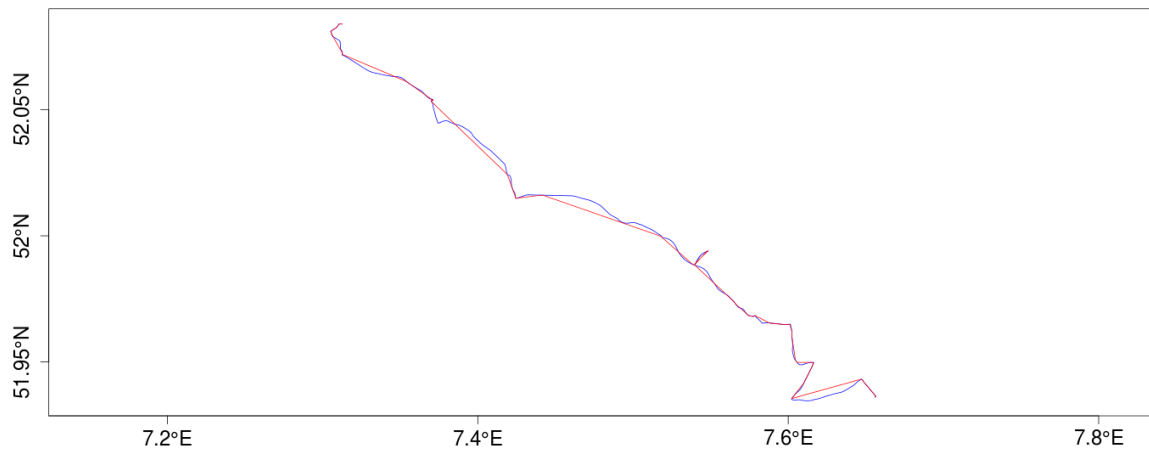
Figure 5.13 - Original trajectory (above) open-window Meratnia-By compression (below), no visible changes.



SOURCE: Author's production.

I also performed a speed filter in the same trajectory that the compression algorithms were tested. In the speed filter the maximum possible speed was set to 20 m/s. As this trajectory does not contain any anomalous points the filter only removed a few points from the real trajectory, making it a bit straighter as seen in figure 5.14.

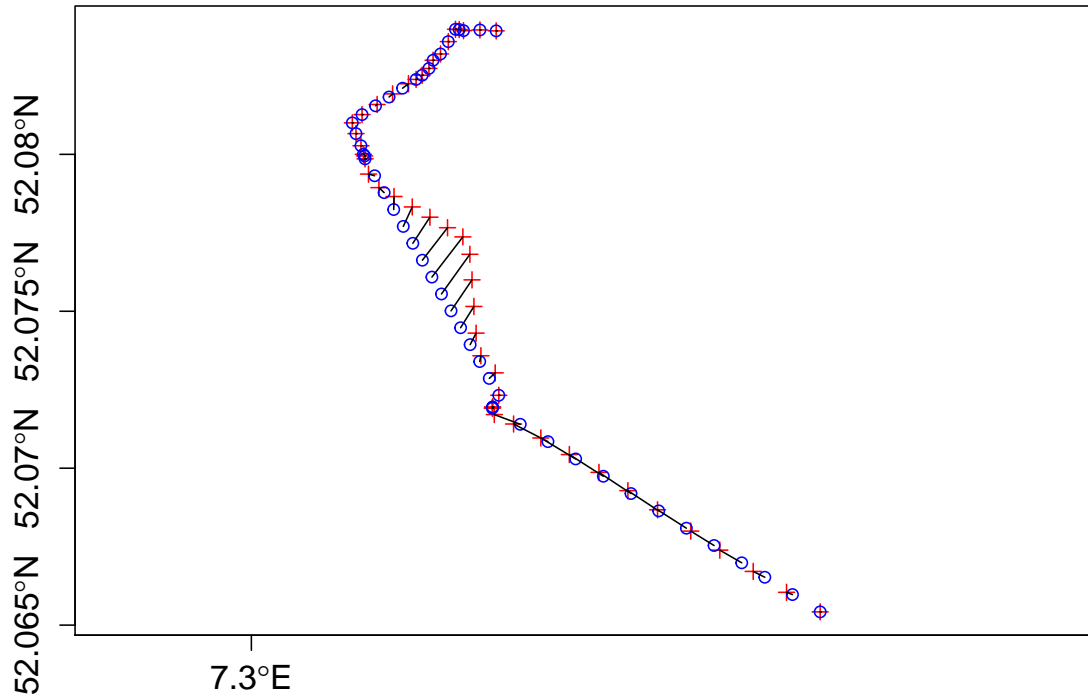
Figure 5.14 - Original trajectory (blue) filtered trajectory (red).



SOURCE: Author's production.

This filtering process shows the importance of knowing what are reasonable speed values for your data. Since the data in figure 5.14 and 5.15 clearly presents some loss of information. This speed filter was tested using different values for the maximum speed, and their errors were calculated. To calculate the error I used the method Compare from the Trajectories package, then I was able to calculate the instantaneous distance between filtered and original trajectory. I used this distance to obtain, accumulated error, maximum error and average error. As the maximum speed was reduced the error was increased as expected. The accumulated error for a maximum speed of 25 m/s was 1.5 m, while this increased to 46 m to 10 m/s and 67 m to 5 m/s. All error measurements followed the same pattern, increasing every time the speed was reduced. The greatest increase happened from the average error from 25 m/s to the 5 m/s one, this error increased 55 times, going from 0.002m to 0.1m.

Figure 5.15 - Trajectory comparison of part of original trajectory (red crosses) and part of filtered trajectory (blue circles).



Comparison uses the method `Compare` from the `Trajectories` package, which uses a linear interpolator.

SOURCE: Author's production.

5.4 Partner discovery

I tested the *partner* algorithms with two different datasets, to validate the theory and test the implementation in different scenarios.

5.4.1 Vessel partner

In the first case study, I used trajectories of over 1000 vessels around the Brazilian coast collected during 6 months in 2008, these trajectories are a subset of the data from section 5.2.

Figure 5.16 - Code to find *Partners* in a DataBase.

```
dataSourceInfo <- DataSourceInfo(user="postgres", password="password",
                                db="vessel_trajectory")

tdsetvessel<- TrajectoryDataSetInfo(dataSetName="vessel_trajectories",
                                    phTimeName="datahora",geomName="ponto",
                                    trajId="traj_id_unique",trajName="",objId="vessel_id")

vesselsstbox<- getIdealSTBoxes(dataSourceInfo,tdsetvessel)

findBigPartnerDB(dataSourceInfo,tdsetvessel,1.5,3600,18000,vesselsstbox,2,
                  "vessels_partners1k5m_1h")
```

The numeric parameters in `findBigPartnerDB` are respectively d_{max} , p_{max} , p_{min} and the number of computer cores used for the operation. The string is the name of the DB in which *partners* will be stored.

SOURCE: Author's production.

On this data set, I executed the algorithm using the following parameters: $d_{max} = 1.5Km$, $p_{max} = 1$ hour, and $p_{min} = 5$ hours. This can be seen in figure 5.16 in which the times are represented in seconds. As a result I obtained over 50 thousand partners. Possibly, these partners are vessels that were fishing together.

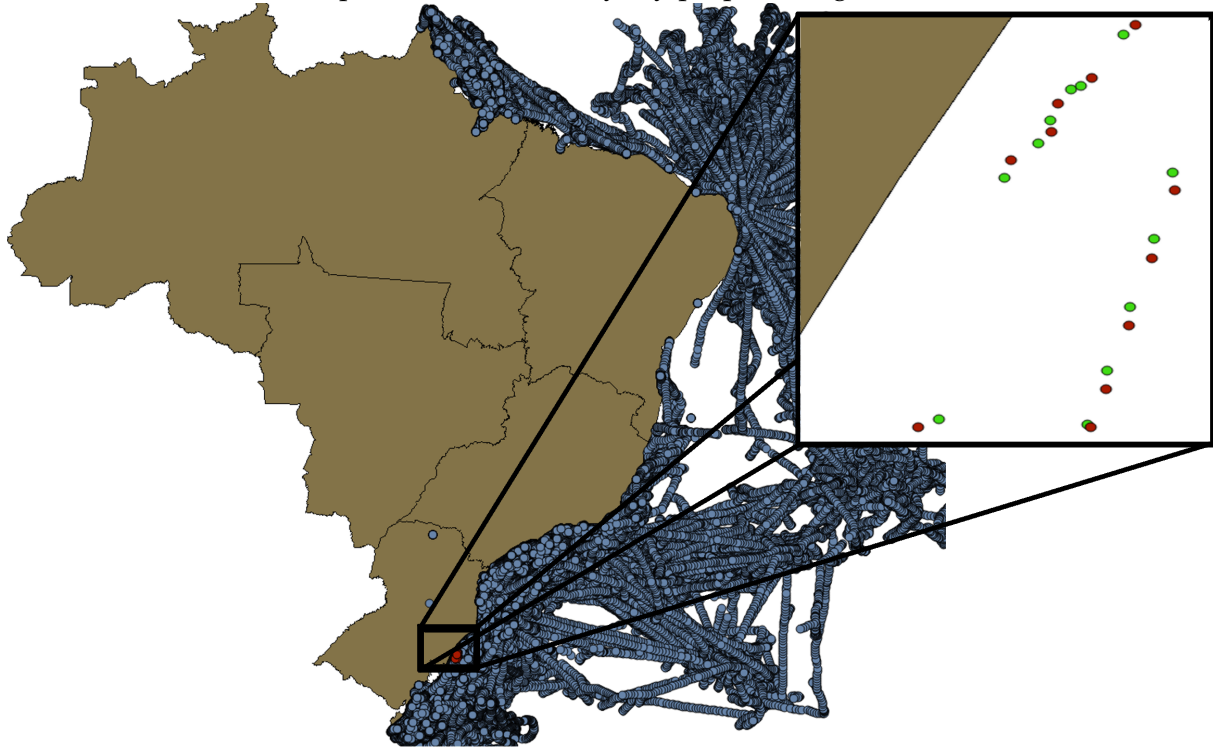
Figure 5.17 shows, on the left side, all vessel trajectories observations and, on the right side, two of these trajectories that are partners identified by the proposed method. On the right side, the region bound by the rectangle is shown in detail, and on it I can see two objects (identified by red and green dots) that are partners. The partners shown in that figure have spent around 11 hours together.

5.4.2 Truck partner

The third dataset consists of 276 trajectories of 50 delivery trucks in Athens, Greece. This dataset was obtained from `chorochronos.org`. Its structure is rather similar to the one of the previous dataset, but since each truck has a different number of trajectories I added an extra column to uniquely identify the pair object-trajectory.

On a first try, I examined the dataset for partners using the same d_{max} value used on the first case study. However, this value is too large for this dataset. Using this value, all trajectories were identified as partners, even the ones going on different directions and different roads. Thus, I chose values that seemed more reasonable for data collected within cities.

Figure 5.17 - Vessels around the Brazilian coast. Blue dots/lines: original trajectories. Red and Green Dots: partners identified by my proposed algorithm.



SOURCE: Author's production

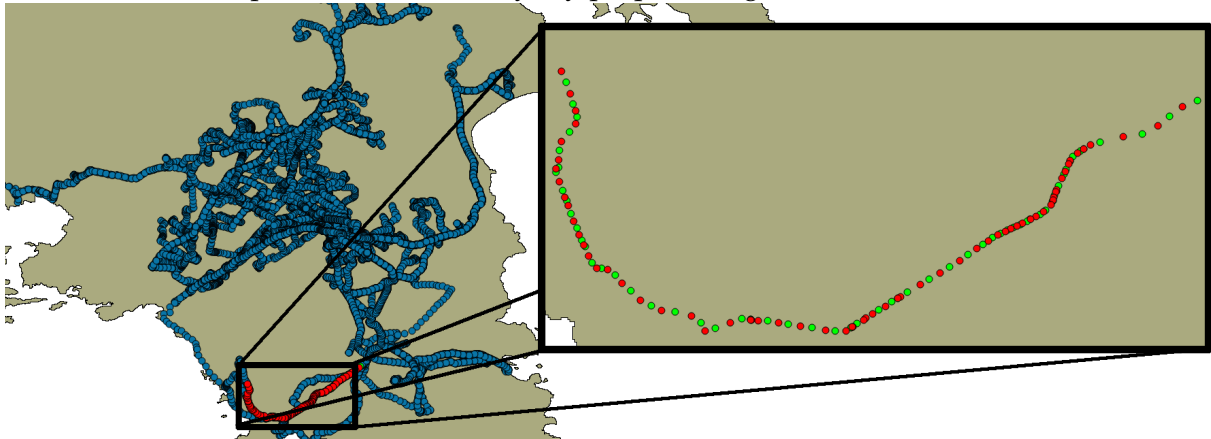
The number of discovered partners depends on the given parameters. I looked for partners that were separated at most by 10, 50 and 100 meters. It is important to reiterate that the analyzed area increases according to d_{max} (Algorithm 3 line 4), which affects the running time of the algorithm.

For $d_{max} = 10m$, the algorithm identified only 31 partners. Since this value of d_{max} is a lot more restrictive, trajectories like the one from Figure 5.18 are not completely identified as partners. For $d_{max} = 50m$, I found 863 partners (one is shown in Figure 5.18). Intuitively the number of partners declined as the acceptable separation distance was shortened, and increased as the acceptable separation distance was extended.

For $d_{max} = 100m$, the algorithm identified 1327 partners. I consider this distance as the more meaningful for the city truck data. Nevertheless, since this value caused an increase on the areas to be analyzed, the algorithm took longer to execute.

This dataset presents an ideal size to test the parallelization capabilities of the algorithm. It is big enough to gain processing time from parallelization, but not too large to be too much time-consuming. To measure performance gains, first I executed the algorithm twice for each distance, using one, two, and three CPU cores.

Figure 5.18 - Truck trajectories in Greece. Blue dots/lines: original trajectories. Red and Green Dots: partners identified by my proposed algorithm.

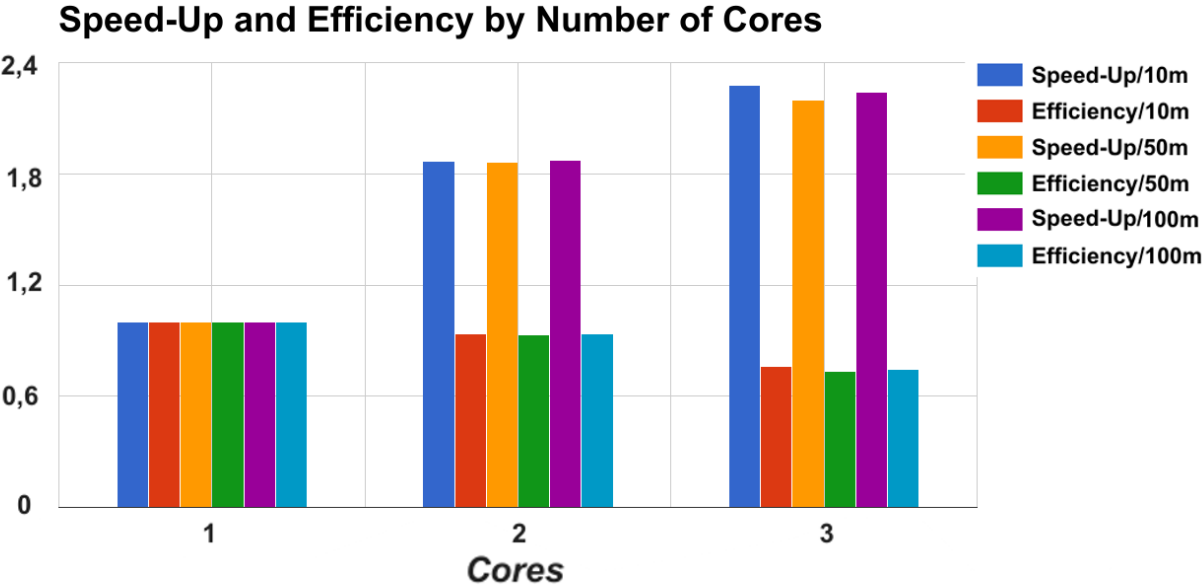


SOURCE: Author's production

Even though each increase in d_{max} made the algorithm more time consuming since a larger area was analyzed, I verified that the speed-up was practically constant for all distances, being 1.87 faster for 2 cores and 2.24 faster for 3 cores, when comparing running time using only one core, this is shown on figure 5.19.

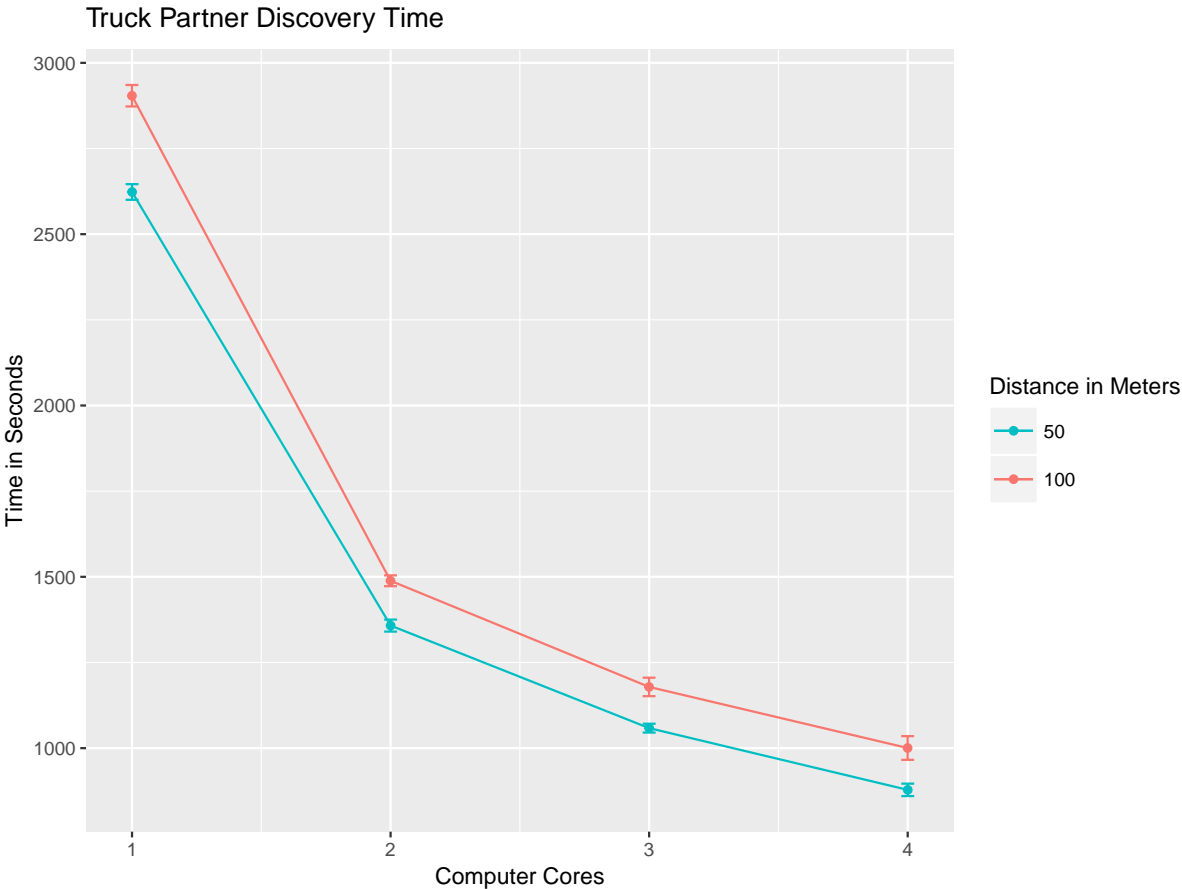
Later I also tested the parallelization on a second machine, equivalent to the other, which was used on all previous case studies, except for the number of cores. This second machine has one extra core. This second test involved running the code 64 times, changing number of cores, d_{max} and p_{max} . There was no difference in the time results when p_{max} was altered. The results on both machines matched, and on figure 5.20, it is possible to see the improvement in speed getting smaller at each extra core being used. The efficiency with 2 cores is 90% whereas with 4 cores it is 75%.

Figure 5.19 - Speed-Up comparison with changing parameters and number of cores.



SOURCE: Author's production

Figure 5.20 - Graph of second time comparison, with one extra core.



SOURCE: Author's production

6 CONCLUSIONS AND FUTURE WORK

In this work, I present existing tools for trajectory analysis, highlight their advantages and disadvantages and point out the need for a high-level programming environment that allows users to access big trajectory data sets and develop new algorithms using them. To meet this demand, I propose a framework for the R environment. I implemented an R package for accessing big trajectory data by parts from different types of sources, as a solution to work in memory limited environments. I discuss existing moving object patterns and propose a new one. Finally I implemented the framework, with the proposed functionalities and my proposed algorithm.

The *Partner* is useful for many applications that need to identify objects that are performing activities jointly, such as marine vessels that are fishing together or trucks that are working in cooperation, as well as objects that are stationary close to others. I test and validate the proposed method through two case studies.

The algorithm can be executed in parallel, allowing the processing of large datasets, but further analyses still need to be done to evaluate when parallelizing this algorithm becomes advantageous.

Further implementations in R should be done in order to compare the efficiency of my proposed algorithm compared to the existing ones.

As future work, I intend to add a new functionality to *TrajDataAccess* package to store results of data analyses in data sources. Besides that, I expect to improve the visualization of big trajectory data sets in *TerraView*, making use of smart sliders, which show trajectories according to their locations in time, instead of showing all points together. The algorithm that calculates the ideal *STboxes* maybe improved to consider more than one dimension at once.

I intend to improve the visualization methods for partners in the *TerraView 5* system and the integration between this system and the R package *TrajDataMining*. Besides that, I want to describe and implement algorithms for identification of bigger groups of partners and to improve the data retrieval of the algorithm, so less data will be required for in-memory analysis.

The concept framework, C++ library and R environment maybe extended to other kinds of data that are not from trajectories, such as those from coverages and other spatiotemporal data.

I make my R implementations available on github at <http://github.com/dvm1607>. To use the TrajDataAccess package, it is necessary to install Terralib 5, available at <http://www.dpi.inpe.br/terralib5>. Some functions may not be available for all operational systems.

REFERENCES

- ALVARES, L. O.; BOGORNY, V.; KUIJPERS, B.; MACEDO, J. A. F. de; MOELANS, B.; VAISMAN, A. A model for enriching trajectories with semantic geographical information. In: ANNUAL ACM INTERNATIONAL SYMPOSIUM ON ADVANCES IN GEOGRAPHIC INFORMATION SYSTEMS, 15th. **Proceedings...** [S.l.]: ACM, 2007. p. 22–29. [1](#), [5](#)
- ALVARES, L. O.; PALMA, A.; OLIVEIRA, G.; BOGORNY, V. Weka-STPM: from trajectory samples to semantic trajectories. In: WORKSHOP DE SOFTWARE LIVRE, WSL, 11th., 2010. **Proceedings...** [S.l.], 2010. v. 10, p. 164–169. [2](#), [12](#)
- ANDRADE, P. R. d.; JR, P. J. R.; FOOK, K. D. Integration of statistics and geographic information systems: the R/Terralib case. In: BRAZILIAN SYMPOSIUM ON GEOINFORMATICS, 7th., 2005, Campos do Jordão, São Paulo, Brazil. **Proceedings...** São José dos Campos: INPE, 2015. p. 139–154. ISSN 2179-4820. [16](#)
- BASKAR, S.; AROCKIAM, L.; CHARLES, S. A systematic approach on data pre-processing in data mining. **COMPUSOFT, An International Journal of Advanced Computer Technology**, v. 2, p. 335, 2013. [7](#)
- BIVAND, R.; KEITT, T.; ROWLINGSON, B. rgdal: Bindings for the geospatial data abstraction library. **R package version 0.8-10**, 2013. [3](#)
- CALENGE, C. The package “adehabitat” for the R software: a tool for the analysis of space and habitat use by animals. **Ecological modelling**, Elsevier, v. 197, n. 3, p. 516–519, 2006. [2](#), [13](#)
- CÂMARA, G.; VINHAS, L.; FERREIRA, K. R.; QUEIROZ, G. R. D.; SOUZA, R. C. M. D.; MONTEIRO, A. M. V.; CARVALHO, M. T. D.; CASANOVA, M. A.; FREITAS, U. M. D. TerraLib: An open source GIS library for large-scale environmental and socio-economic applications. In: **Open source approaches in spatial data handling**. [S.l.]: Springer, 2008. p. 247–270. [3](#), [13](#)
- CONWAY, J.; EDDERBUETTEL, D.; NISHIYAMA, T.; PRAYAGA, S. K.; TIFFIN, N. RPostgreSQL: R interface to the PostgreSQL database system. **R package version 0.3-2**, 2012. [3](#)
- DOUGLAS, D. H.; PEUCKER, T. K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. **Cartographica: The International Journal for Geographic Information and Geovisualization**, University of Toronto Press, v. 10, n. 2, p. 112–122, 1973. [8](#), [25](#)

- EDDELBUETTEL, D.; FRANÇOIS, R.; ALLAIRE, J.; CHAMBERS, J.; BATES, D.; USHEY, K. Rcpp: Seamless R and C++ integration. **Journal of Statistical Software**, v. 40, n. 8, p. 1–18, 2011. 20
- ERWIG, M.; GU, R. H.; SCHNEIDER, M.; VAZIRGIANNIS, M. et al. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. **GeoInformatica**, Springer, v. 3, n. 3, p. 269–296, 1999. 1
- ETIENNE, L. **Motifs spatio-temporels de trajectoires d’objets mobiles, de l’extraction à la détection de comportements inhabituels**. Application au trafic maritime. PhD Thesis (PhD) — Université de Bretagne occidentale-Brest, 2011. 2, 5, 6, 7, 8, 25
- FAN, Q.; ZHANG, D.; WU, H.; TAN, K.-L. A general and parallel platform for mining co-movement patterns over large-scale trajectories. **Proceedings of the VLDB Endowment**, v. 10, n. 4, 2016. 10, 11
- FAYYAD, U.; PIATETSKY-SHAPIO, G.; SMYTH, P. From data mining to knowledge discovery in databases. **AI Magazine**, v. 17, p. 37–54, 1996. 5, 6
- FERREIRA, K. R.; CAMARA, G.; MONTEIRO, A. M. V. An algebra for spatiotemporal data: From observations to events. **Transactions in GIS**, Wiley Online Library, v. 18, n. 2, p. 253–269, 2014. 14, 15, 20
- FERREIRA, K. R.; OLIVEIRA, A. G. de; MONTEIRO, A. M. V.; ALMEIDA, D. B. de. Temporal GIS and spatiotemporal data sources. In: BRAZILIAN SYMPOSIUM ON GEOINFORMATICS, 16th., 2015, Campos do Jordão, São Paulo, Brazil. **Proceedings...** São José dos Campos: INPE, 2015. p. 1–13. ISSN 2179-4820. Available from: <<http://urlib.net/8JMKD3MGPDW34P/3KP2RBP>>. 3, 13, 15, 20
- FERREIRA, K. R.; VINHAS, L.; MONTEIRO, A. M. V.; CÂMARA, G. Moving objects and spatial data sources. **Revista Brasileira de Cartografia**, v. 64, n. 4, 2013. 1
- GEERTS, F. Moving objects and their equations of motion. In: **Constraint Databases**. [S.l.]: Springer, 2004. p. 40–51. 5
- GUDMUNDSSON, J.; KREVELD, M. van. Computing longest duration flocks in trajectory data. In: ANNUAL ACM INTERNATIONAL SYMPOSIUM ON ADVANCES IN GEOGRAPHIC INFORMATION SYSTEMS, 14th. **Proceedings...** [S.l.]: ACM, 2006. p. 35–42. 10
- JEUNG, H.; YIU, M. L.; ZHOU, X.; JENSEN, C. S.; SHEN, H. T. Discovery of convoys in trajectory databases. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 1, n. 1, p. 1068–1080, 2008. 1, 10

- KANE, M. J.; EMERSON, J.; WESTON, S. Scalable strategies for computing with massive data. **Journal of Statistical Software**, v. 55, n. 14, p. 1–19, 2013. 3, 12
- KLUS, B.; PEBESMA, E. **Analysing Trajectory Data in R**. [S.l.], 2015. Available from: <<http://cran.wustl.edu/web/packages/trajectories/vignettes/tracks.pdf>>. 2, 13, 19
- KUMAR, V.; GRAMA, A.; GUPTA, A.; KARYPIS, G. **Introduction to Parallel Computing**. 2nd. [S.l.]: Addison Wesley, 2003. 32
- LEA, D. A Java fork/join framework. In: ACM 2000 CONFERENCE ON JAVA GRANDE, San Francisco, California. **Proceedings...** [S.l.], 2000. p. 36–43. 33
- LI, Y.; BAILEY, J.; KULIK, L. Efficient mining of platoon patterns in trajectory databases. **Data & Knowledge Engineering**, Elsevier, v. 100, p. 167–187, 2015. 11
- LI, Z.; DING, B.; HAN, J.; KAYS, R.; NYE, P. Mining periodic behaviors for moving objects. In: ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, 16th. **Proceedings...** [S.l.]: ACM, 2010. p. 1099–1108. 10, 11
- LOY, A.; BOGORNY, V.; RENSO, C.; ALVARES, L. O. Um algoritmo para identificar padrões comportamentais do tipo avoidance em trajetórias de objetos móveis. **on Geoinformatics**, p. 158, 2010. 1
- MERATNIA, N.; BY, R. A. **Spatiotemporal Compression Techniques for Moving Point Objects**. Heraklion, Crete, Greece: International Conference on Extending Database Technology, 2004. 765–782 p. 7, 8, 9, 25, 44
- MÜLLNER, D. et al. fastcluster: Fast hierarchical, agglomerative clustering routines for R and Python. **Journal of Statistical Software**, Foundation for Open Access Statistics, v. 53, n. 9, p. 1–18, 2013. 2
- PALMA, A. T.; BOGORNY, V.; KUIJPERS, B.; ALVARES, L. O. **A Clustering-based Approach for Discovering Interesting Places in Trajectories**. [S.l.]: ACMSAC, 2008. 863–868 p. 9, 25
- PEBESMA, E. spacetime: Spatio-temporal data in R. **Journal of Statistical Software**, v. 51, n. 7, p. 1–30, 2012. 2
- R Development Core Team. **R: A Language and Environment for Statistical Computing**. Vienna, Austria, 2011. ISBN 3-900051-07-0. Available from: <<http://www.R-project.org>>. 2, 12, 13

- RENZO, C.; SPACCAPIETRA, S.; ZIMÁNYI, E. **Mobility Data**. [S.l.]: Cambridge University Press, 2013. [1](#), [2](#), [5](#), [12](#)
- ROCHA, J. A.; TIMES, V. C.; OLIVEIRA, G.; ALVARES, L. O.; BOGORNY, V. **DB-SMoT: A direction-based spatio-temporal clustering method**. [S.l.]: 5th IEEE International Conference Intelligent Systems, 2010. 114–119 p. [9](#), [25](#)
- SPACCAPIETRA, S.; PARENT, C.; DAMIANI, M. L.; MACEDO, J. A. de; PORTO, F.; VANGENOT, C. A conceptual view on trajectories. **Data & knowledge engineering**, Elsevier, v. 65, n. 1, p. 126–146, 2008. [1](#), [9](#)
- TANAKA, P. S.; VIEIRA, M. R.; KASTER, D. S. Efficient algorithms to discover flock patterns in trajectories. In: BRAZILIAN SYMPOSIUM ON GEOINFORMATICS, 16th., 2015, Campos do Jordão, São Paulo, Brazil. **Proceedings...** São José dos Campos: INPE, 2015. p. 56–67. ISSN 2179-4820. [10](#)
- TANG, L.-A.; ZHENG, Y.; YUAN, J.; HAN, J.; LEUNG, A.; HUNG, C.-C.; PENG, W.-C. On discovery of traveling companions from streaming trajectories. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 28th. **Proceedings...** [S.l.]: IEEE, 2012. p. 186–197. [10](#), [11](#)
- TOOHEY, K.; DUCKHAM, M. Trajectory similarity measures. **SIGSPATIAL Special**, ACM, v. 7, n. 1, p. 43–50, 2015. [2](#), [13](#)
- VAUGHAN, J.; WHYATT, D.; BROOKES, G. M. d. A parallel implementation of the Douglas Peucker line simplification algorithm. **SOFTWARE PRACTICE AND EXPERIENCE**, v. 21, p. 331–336, 1991. [8](#)
- VIEIRA, M. R.; BAKALOV, P.; TSOTRAS, V. J. On-line discovery of flock patterns in spatio-temporal data. In: ACM SIGSPATIAL INTERNATIONAL CONFERENCE ON ADVANCES IN GEOGRAPHIC INFORMATION SYSTEMS, 17th. **Proceedings...** [S.l.]: ACM, 2009. p. 286–295. [1](#), [10](#)
- WANG, Y.; LIM, E.-P.; HWANG, S.-Y. Efficient mining of group patterns from user movement data. **Data & Knowledge Engineering**, Elsevier, v. 57, n. 3, p. 240–282, 2006. [10](#)
- ZHENG, K.; ZHENG, Y.; YUAN, N. J.; SHANG, S. On discovery of gathering patterns from trajectories. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 29th. **Proceedings...** [S.l.]: IEEE, 2013. p. 242–253. [10](#), [11](#)
- ZHENG, K.; ZHENG, Y.; YUAN, N. J.; SHANG, S.; ZHOU, X. Online discovery of gathering patterns over trajectories. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 26, n. 8, p. 1974–1988, 2014. [10](#), [11](#)

ZHENG, Y. Trajectory data mining: an overview. **ACM Transactions on Intelligent Systems and Technology (TIST)**, ACM, v. 6, n. 3, p. 29, 2015. [1](#), [10](#), [11](#)

APPENDIX A - LIST OF FUNCTIONS OF TRAJDATAACCESS PACKAGE

getTrajectory	Method, that given a DataSourceInfo and a TrajectoryDataSetInfo, returns trajectories
getTrajectoryByTrack	Method, that given a DataSourceInfo and a TrajectoryDataSetInfo, returns trajectories that intersect the STBox of a given Track
getTrajectoryBySTBox	Method, that given a DataSourceInfo and a TrajectoryDataSetInfo, returns trajectories that intersect a given STBox
getTrajectoryByID	Method, that given a DataSourceInfo, a TrajectoryDataSetInfo and a Trajectory ID, returns the trajectory whose ID matches the given Trajectory ID.
getNeededDivisions	Method, that given a DataSourceInfo and a TrajectoryDataSetInfo, returns the adequate number of divisions for the dataset to be loaded in R
getIdealSTBoxes	Method, that given a DataSourceInfo and a TrajectoryDataSetInfo, returns the ideal STBoxes for the data set to be loaded in R
findBigPartnerDB	Method, that given a DataSourceInfo, a TrajectoryDataSetInfo, d_{max} , p_{max} and p_{min} returns the <i>partners</i> in the DataSet

SOURCE: Author's production.

APPENDIX B - LIST OF FUNCTIONS OF TRAJDATAMINING PACKAGE

directionCluster	Method, that given a Track and maximum change parameter, returns regions where direction changed more than the defined parameter
douglasPeucker	Method that reduces a trajectory spatially
partner	Method to identify if two trajectories are partners
owMeratniaBy	Method that reduces trajectories spatiotemporally
owMeratniaByCollection	Method that reduces a set of trajectories spatiotemporally
sendPartner	Method that sends found partners to a PostGIS database
speedCluster	Method, that given a Track and minimum speed parameter, returns regions where speed was lower than the defined parameter
speedFilter	Method, that given a Track and maximum speed parameter, removes regions where speed was higher than the defined parameter

SOURCE: Author's production.

