UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

TICIANA LINHARES COELHO DA SILVA

ONLINE CLUSTERING OF TRAJECTORY DATA STREAM

FORTALEZA

2016

TICIANA LINHARES COELHO DA SILVA

ONLINE CLUSTERING OF TRAJECTORY DATA STREAM

Tese apresentada ao Curso de Doutorado em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Ciência da Computação

Orientador: Prof. Dr. José Antônio Fernandes de Macêdo

Co-Orientador: Prof. Dra. Karine Bennis Zeitouni

FORTALEZA

2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

C621o Coelho da Silva, Ticiana Linhares.
Online Clustering of Trajectory Data Stream / Ticiana Linhares Coelho da Silva. – 2016.
107 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação , Fortaleza, 2016.
Orientação: Prof. Dr. José Antônio Fernandes de Macêdo.
Coorientação: Prof. Dr. Karine Bennis Zeitouni.

1. Clusterização. 2. Trajetória. 3. Padrão de Mobilidade. I. Título.

CDD 005

TICIANA LINHARES COELHO DA SILVA

ONLINE CLUSTERING OF TRAJECTORY DATA STREAM

Tese apresentada ao Curso de Doutorado em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Ciência da Computação

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. José Antônio Fernandes de Macêdo (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dra. Karine Bennis Zeitouni
(Co-Orientador)
Universidade de Versailles Saint Quentin en Yvelines (UVSQ)

Prof. Dr. João Paulo Pordeus Gomes
Universidade Federal do Ceará (UFC)

Prof. Dr. Marco Antônio Casanova
Pontifícia Universidade Católica do Rio de Janeiro (PUC)

Prof. Dr. Francesco Corona
Universidade Federal do Ceará (UFC)

Prof. Dr. João Eduardo Ferreira
Universidade de São Paulo (USP)

À Deus, por cuidar sempre de mim.

AGRADECIMENTOS

This acknowledgement is to express my thankfulness to all who helped me over these years.

First, I would like to thank our God for giving me the strength to achieve my dream: become a doctor in Computer Science. To my parents, Paulo Sérgio Oliveira Coelho da Silva and Silvia Helena Linhares Coelho da Silva, for their moral and education support in my whole life. To my brother Paulo Junior who I love so much and who always made me happy when I got stressed. To my fiance Nicolas who faced my dream as his dream and for his endless love.

My first deepest gratitude is to my advisor, Prof. Jose Macedo. It has been an honor for me to be one of his PhD students. I really appreciate all his great contribution in my thesis, encouragement and all the opportunities I had so far.

Thanks to my co-advisor Prof. Karine Zeitouni who welcomed me in Universite de Versailles Saint Quentin en Yvelines (UVSQ) where I spent one year of my thesis very focused. It was an excellent and conscientious support. She was always available to advice me. I would like to also thank Prof. Marco Casanova who collaborated over these years, sending me very good remarks and notes to support this thesis.

To all my friends from ARIDa: Samara, Toni, Régis, Bruno, Flávio, Thiago, Lívia, Gustavo, Camila, Victor, Davi, Peres, Erick, Igo and Francesco; To my friends from InfoLab in Greece: Despina, Panos, Marios for all very good time that we spent in Greece and Brazil, thanks also to Prof. Yannis Theodoridis, Prof. Nikos Pelekis and Prof. Kostas Patroumpas who gave me many advices during my stage there; To my friends from UVSQ in France: Hanane Ouksili, Dai Hai, Twiza, Mariem, Ali Masri, Raef Mousheimish, Amino, Naila Bouchemal, Mary, Nicoleta Preda, Ameni, Tarek and Nathan. Also to my very close friends Márcio, Atslands, Hylleia, Cristiana, Amanda, Rafaela, Marina, Isabel, Arthur, Valterfi, Esther, Robson and Lara. I am very grateful for all their cheerfulness and friendship.

To my colleagues and students of UFC Quixadá Campus where I teach, thanks to let me take one year off in order to dedicate my time only to work on my thesis. To the professors of ARIDa research group who always motivated me over these years.

And lastly to all professors and colleagues that I had collaborated for all their full cooperation that made them a big part of this work.

"Uma criança, um professor, um livro e uma caneta podem mudar o mundo. Educação é solução."

(Malala)

RESUMO

A mineração de dados de trajetória permitem caracterizar o comportamento de movimento (isto é, congestionamento, *flocks, swarms, leadership*, entre outros padrões de movimento), impulsionando novas aplicações e serviços. O rastreamento de objetos móveis se torna onipresente em muitas aplicações, o que gera grande interesse na análise de dados de trajetória e na mineração destes dados. A maioria das abordagens permite detectar padrões de movimento em dados históricos de objetos, mas não padrões atuais. Abordagens recentes para clusterização online de objetos móveis se restringem a analisar posições instantâneas. Dessa forma, estes trabalhos não conseguem capturar o comportamento dos objetos em movimento ao longo do tempo. Ao monitorar continuamente as subtrajetórias de objetos móveis em intervalos de tempo, ao invés de apenas a última posição, é possível obter uma visão sobre o comportamento atual e potencialmente detectar padrões de mobilidade em tempo real.

A análise em tempo real dos dados de mobilidade pode oferecer conhecimento para entender melhor a dinâmica da cidade em curso, bem como a detecção de irregularidades e anomalias à medida que acontecem; Este estudo é relevante para tomada de decisão. Entre os possíveis padrões, nesta tese consideramos principalmente o agrupamento (clusterização) de sub-trajetórias e a evolução do movimento dos objetos. Descobrir esses padrões pode ajudar na re-engenharia do tráfego de grandes cidades, ou para detectar prontamente eventos (por exemplo, acidentes de carro, passeatas, entre outros) à medida que eles acontecem.

Na primeira linha de investigação desta tese, abordamos o problema de descobrir e manter os *clusters* baseados em densidade utilizando *streams* de trajetórias no Espaço Euclidiano, levando em consideração que a maioria dos objetos em movimento muda de posição ao longo do tempo. Dessa forma, esta tese propõe o framework CUTiS (*ClUstering Trajectory Stream*), um algoritmo incremental para resolver este problema. CUTiS é capaz de monitorar a evolução dos padrões de movimento (clusters), bem como a adesão dos objetos em movimento aos padrões já existentes. Nossos experimentos foram conduzidos em dois conjuntos de dados reais e os experimentos mostram a eficiência e a eficácia do nosso método comparando a dois concorrentes DBSCAN (ESTER *et al.*, 1996) e TraClus (LEE *et al.*, 2007).

Como segunda linha de pesquisa, esta tese teve como objetivo melhorar a eficiência do algoritmo CUTiS. Desta forma, foi proposto uma estrutura de indexação para dados de sub-trajetória com base em *space filling curve*. Esta abordagem tem a propriedade de mapear um espaço multidimensional para um espaço unidimensional tal que, para dois objetos que estão próximos

no espaço original, existe uma alta probabilidade de que eles fiquem próximos no espaço alvo mapeado (THAT *et al.*, 2016). Essa propriedade foi utilizada para otimizar *range queries* de uma sub-trajetória de um objeto no algoritmo de clusterização incremental. Nossos experimentos foram conduzidos em um conjunto de dados reais e eles mostram a eficiência e a eficácia do nosso método em comparação com a nossa proposta anterior CUTiS, e as abordagens DBSCAN e TraClus.

Como terceira linha, investigamos o mesmo problema de descoberta e manutenção de clusters de sub-trajetórias em uma rede rodoviária, já que muitos objetos em movimento se movem na rede rodoviária em aplicações reais. Dessa forma, foi proposto como solução o Net-CUTiS, um algoritmo de clusterização incremental para sub-trajetórias de objetos com restrição de movimento em rede rodoviária. A eficiência e a eficácia do Net-CUTiS foram comparadas usando um conjunto de dados real com a abordagem NETSCAN (KHARRAT *et al.*, 2008) e DBSCAN (ESTER *et al.*, 1996).

Palavras-chave: Clusterização. Trajetória. Padrão de Mobilidade.

ABSTRACT

Mining trajectory patterns allows characterizing movement behavior (i.e. congestion, flocks, swarms, leadership, among others), which leverages new applications and services. Movement tracking becomes ubiquitous in many applications, which raises great interests in trajectory data analysis and mining. Most existing approaches allow characterizing the past movements of the objects but not current patterns, because they use only historical trajectory data. Recent approaches for online clustering of moving objects location are restricted to instantaneous positions. Subsequently, they fail to capture moving objects' behavior over time. By continuously tracking moving objects' sub-trajectories at each time window, rather than just the last position, it becomes possible to gain insight on the current behavior, and potentially detect mobility patterns in real time.

Real-time analysis of mobility data may offer novel tools to better understand ongoing city dynamics, as well as the detection of regularities and anomalies as they happen; all in all, this can represent an invaluable tool when tackling decision-making tasks. Among the possible patterns, in this thesis we mainly consider (sub)-trajectory clustering and its evolution. Discovering such patterns may help to re-engineer effectively the traffic within a city, or to promptly detect events at the city level (e.g., car accidents) as they happen.

In the first line of investigation we tackle the problem of discovering and maintaining the density based clusters in trajectory data streams in Euclidean Space, despite the fact that most moving objects change their position over time. We propose CUTiS, an incremental algorithm to solve this problem, while tracking the evolution of the clusters as well as the membership of the moving objects to the clusters. Our experiments were conducted on two real datasets and the experiments show the efficiency and the effectiveness of our method comparing to two competitors DBSCAN (ESTER *et al.*, 1996) and TraClus (LEE *et al.*, 2007).

As a second line of research, we aim at improving the efficiency of the CUTiS algorithm. In this way, we propose an indexing structure for sub-trajectory data based on a space-filling curve. This approach has the property of mapping a multidimensional space to one-dimensional space such that, for two objects that are close in the original space, there is a high probability that they will be close in the mapped target space (THAT *et al.*, 2016). We take advantage of this property to optimize range queries from a moving object sub-trajectory on the incremental clustering algorithm. Our experiments were conducted on a real data set and they show the efficiency

and the effectiveness of our method compared to our previous proposed CUTiS, DBSCAN and TraClus.

As a third line, we investigate the same problem of sub-trajectory clustering discovery and maintenance on a Road Network since many moving objects move on the road network in real applications. We propose Net-CUTiS an incremental clustering algorithm for road network constraint movement. The efficiency and effectiveness of Net-CUTiS were compared using a real dataset with NETSCAN (KHARRAT *et al.*, 2008) and DBSCAN (ESTER *et al.*, 1996).

Keywords: Trajectory Clustering. Road Network. Mobility Pattern.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUCTION

1.1 Motivation

The huge volume of collected trajectories opens new opportunities for discovering hidden mobility patterns. These patterns allow characterizing individual mobility as well as groups sharing similar trajectories for a certain time interval. Usually, this analysis is done off-line by applying data analysis and mining techniques on the previously collected data. This allows characterizing the past movements of the objects but not the current mobility patterns. Nowadays, many services involve tracking moving objects (e.g., persons, vehicles, animals) to report their trajectory continuously (e.g., every second or every minute). Analyzing these data while they are generated may bring a real added-value in the comprehension of the city dynamics, and the detection of regularities as well as anomalies, which is essential for decision-making. Among these patterns, we mainly consider in this thesis the sub-trajectory clustering and its evolution. Such discovery may help the search for effective re-engineering of traffic, or dynamically detecting events or incidents at a city level.

Tracking applications need to deal with the incremental nature of spatio-temporal data. Indeed, data arrive rapidly in a short period of time, and the size of the data set keeps growing as time goes by. The exploration of the entire data stream might not be very useful since the information to be extracted may be outdated. Analyzing the data while it arrives can provide much better comprehension of current cluster patterns and their evolution over consecutive periods of time. By current cluster patterns, we mean that the groups of moving objects having similar sub-trajectories over the last time window. Thus, by tracking sub-trajectories rather than just the last position, it becomes possible to gain insight on the current behavior, and potentially detect suspicious behaviors or remarkable events the moment they happen. However, finding clusters in these data streams in quasi real time is challenging because all tracked moving objects may change their positions every time, and clusters may also change accordingly. Furthermore, new moving objects may appear, and others may stop and disappear. These changes may affect the formation of clusters. The exploration of the stream over different intervals of time (time windows) can provide the users much better comprehension of the evolving behavior of the clusters. However, executing the clustering algorithm from scratch every time window may lead to a high cost since we have a large volume of streaming data.

Approaches exist for online clustering of the positions of moving objects, but they

are restricted to instantaneous positions. Furthermore, they fail to capture the moving objects displacement behavior along time. The majority of existing mobility pattern approaches, as trajectory clustering, flocks, convoys, swarms, gathering (ZHENG, 2015), analyze historical sub-trajectory data; no solution exists for clustering and maintaining clusters of sub-trajectories in real time. That is why we claim that our study is relevant.

In this thesis, we address the problem of online discovery of mobility patterns and their evolution by tracking the sub-trajectories of moving objects (for both scenarios, in Euclidean and Road Network Space) at each time window. We consider that moving objects may change their positions continuously; new moving objects appear and others disappear from the system during an observed time interval. In the next section, we describe the research problems addressed by this thesis.

1.2 Research Problems

The traditional trajectory model in the literature captures the movement of an object in geographical space over some period of time. Nowadays, the movement of people or vehicles within a given area can be observed from the digital traces left behind by the personal or vehicular mobile devices, and collected by the wireless network infrastructures. For instance, mobile phones leave positioning logs, which specify their localization, or cell, at each moment they are connected to the GSM network; analogously, GPS-equipped portable devices can record their latitude-longitude position at each moment they are exposed to a GPS satellite, and can transmit their trajectories to a collecting server (GIANNOTTI *et al.*, 2007).

The pervasiveness of ubiquitous technologies guarantees an increasing availability of large amounts of spatio-temporal data. Therefore, there is an opportunity – and a challenge – to discover online, from these trajectories, spatio-temporal patterns that convey useful knowledge. For example, consider a private company's transportation system that tags and tracks its vehicles as they move around. The devices report the trajectories on a continual basis (e.g., every minute, every hour). With such data, it is possible to support drivers' decisions related to displacement through the city in real time (i.e., traffic jams or accidents). Such discovery may help the search for effective re-engineering of traffic solutions in the context of smart cities. A second issue is that the maintenance of these patterns over continuous data streams needs to be done in an efficient way, which is challenging since all tracked moving objects may change their positions every time they report, and the patterns may also change accordingly. Moreover, new moving objects may

appear and others may stop and disappear. These operations may change the previous patterns found.

However, to execute mining algorithms from scratch every time window may lead to a high cost since we have large volumes of streaming data. It is not only the performance which is affected by applying mining algorithms from scratch; moreover, it is the semantic information about the patterns and how these patterns evolve according to small changes in the movement of objects. Typical evolution patterns include: (1) a new pattern appears, (2) a pattern disappears, (3) an existing pattern splits into different patterns, (4) two or more existing patterns merge into one pattern, and (5) an existing pattern survives as time goes by (SPILIOPOULOU *et al.*, 2006). For instance, imagine a traffic jam as a sequence of merges, or as an evolution of groups of objects which keep their movement similar in space and time. By tracking the small changes of these moving objects, we can capture in (quasi) real time when a traffic jam occurs and how long it is going to last. Then, in this thesis, we continuously track the moving objects in small time intervals (also called time windows) in order to capture how the small changes in the movement behavior of objects affect the patterns and how they evolve in time.

For trajectory patterns, there exist many works to handle online trajectory clustering. These approaches consider the movement in Euclidean space, while showing reasonable performance for clustering unconstrained trajectories (e.g. the free movement of animals through a forest or the movement of hurricanes across an ocean). However, they do not take into account that all tracked moving objects may change their sub-trajectory at each time interval; tracking may start for new moving objects and may stop for others. Furthermore, trajectory clustering should take into account the traffic locality characterized by the spatial constraints of the underlying network as well. The distance between two moving objects on a road network is equivalent to the *length* of the *shortest path* connecting them, while in Euclidean spaces this is simply given by the length of the segment connecting the objects. In this way, it is more challenging to discover online trajectory clustering on a road network, since the distance computation can be a bottleneck.

We aim to support potential applications which need to track moving objects, such as a company which needs to track how its taxi fleet is moving around the city. Furthermore, we need to detect, for a shipping company, when a deliverer is deviating from the common path and how long it takes. This thesis solution offers a framework which is able to provide semantic information online by capturing the patterns and their evolutions in time for objects that move in

Euclidean or on Road Network space.

Considering these observations and the objective of this thesis, the main challenges which this thesis addresses are:

1. (**Challenge 1**) Trajectory Clusters have applications in monitoring hot traffic paths, analyzing people's movement, discovering common hurricane paths, and understanding city dynamics. For these types of applications, there is a need to track trajectory clusters online. How can we discover spatio-temporal patterns online as sub-trajectory clustering for unconstrained trajectories?
2. (**Challenge 2**) In this thesis, we consider that new moving objects might start being tracked and others may stop and disappear. Moreover, the tracked objects might update their position as time goes by. How can we efficiently maintain the spatio-temporal patterns over a continuous trajectory data stream?
3. (**Challenge 3**) As the tracked moving objects evolve as time goes by, the mobility patterns also evolve. How can we detect the frequent evolution (or a sequence of evolutions) of spatio-temporal patterns over a continuous trajectory data stream?
4. (**Challenge 4**) As the majority of objects have constrained movement in a real application and the traditional network distance computations (as shortest path distance) are costly, how can we discover online, and in an efficient way, trajectory clusters on a road network?

In the next section, we discuss the main contribution of this thesis based on the problems presented.

1.3 Thesis Contribution

The main contributions of this thesis are the following:

- Since typical cluster evolution patterns include appear, disappear, expand, shrink, split, merge, and survive (SPILIOPOULOU *et al.*, 2006), we propose a model for cluster evolution patterns which captures the evolution and interaction of these clusters over time, and the frequent sequence evolution patterns of the moving object data. Discovering frequent patterns is not reduced to counting how many times a pattern occurs in the dataset. It is computationally costly to calculate the frequency for every possible sequence of cluster evolution in the data set. The results were published in (SILVA *et al.*, 2014b). This proposal refers to our **Challenge 3**.
- We propose CUTiS (standing for ClUstering Trajectory Stream), an incremental clustering algorithm for trajectory data streams with these main characteristics: (1) most of the objects change their sub-trajectories data at each time interval, (2) new moving objects appear, and (3) others disappear from the system. Because we do clustering for each time window rather than for the whole trajectory, the clusters we find are actually sub-trajectory clusters. We define a new structure, called micro-group, to represent the relationships among moving objects. A micro-group is a small and dense group of moving objects around a representative trajectory (which tries to represent the overall behavior of the group) at each time window. This adapts to many situations such as families staying together or persons sharing the same trip using public transport. Then, micro-groups may evolve, e.g., merge or split in the next time period. In our experiments, the maintenance of the structure of micro-groups presents less computational cost than running the clustering algorithm from scratch at each time window. The results were published in (SILVA *et al.*, 2016b; SILVA *et al.*, 2016a). This proposal refers to our **Challenges 1 and 2**.
- We designed and developed a demo which implements the framework CUTiS for three mobility patterns: (1)*Micro-group Sub-trajectory*, (2) *Density Based Sub-trajectory Clusters* and (3) *Trajectory Flocks*. In this case, a trajectory flock is a subset of moving object trajectories (with at least m moving objects) that stay together (according to a radius r) in the same micro-group for more than a time interval threshold (k time windows). The major difference between our flocks and micro-groups is the time interval constraint in the first. At each time window, the demo enables the user to see the micro-groups' movement (as well as its evolution) and the mobility patterns discovered. The experiments allow the

comparison with many approaches in the state of the art, from both perspectives of quality and performance. The results were published in (SILVA *et al.*, 2016d). This proposal refers to our **Challenges 1 and 2**.

- We developed an indexing structure for sub-trajectory data based on a space-filling curve which is a curve whose range contains the entire 2-dimensional unit square (or more generally an n-dimensional unit hypercube). It has the property of mapping a multidimensional space to one-dimensional space such that, for two objects that are close in the original space, there is a high probability that they will be close in the mapped target space (THAT *et al.*, 2016). We take advantage of this property to optimize the range queries from a moving object sub-trajectory on the incremental clustering algorithm, since during the clustering processing we need to retrieve the trajectories' neighbors of a moving object trajectory. In this way, we develop CUTiS*. The proposed index improves the performance without losing quality in the clusters' results as we show in our experiments conducted on a real dataset. The results were published in (SILVA *et al.*, 2016c). This proposal refers to our **Challenges 1 and 2**.
- We proposed and developed extended solution to scenarios with spatio-temporal constraints, developing the methods to discover the clusters for road network constraint trajectories. We proposed two contributions in this scenario: (1) a spatio-temporal distance function for trajectory data considering its movement on a road network, and (2) an incremental clustering approach for road network constraint trajectory data. We call this approach Net-CUTiS. This proposal refers to our **Challenge 4**.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2 we discuss the state of the art related to this thesis. In particular, we present works related to trajectory data management. Later, the works that proposed approaches to find mobility patterns such as flocks, swarms, and convoys, among others, and trajectory clustering in Euclidian and Road Network constraint movement are presented.

Chapter 3 describes our naive solution to detect cluster evolution patterns and how to detect a frequent sequence of evolutions. In Chapter 4 we present our framework CUTiS for Euclidean Space and in Chapter 5 Net-CUTiS for Road Network constraint movement. In both chapters, the evaluation of our approach is presented for real datasets by evaluating the efficiency and effectiveness compared with our competitors. The framework CUTiS presents three main contributions:

- A novel definition of micro-group that we believe enables the computation of interesting patterns.
- An incremental algorithm to maintain micro-groups and to capture their evolution patterns on trajectory data streams.
- A sub-trajectory clustering algorithm based on time, space, and angular distance functions.

Finally, Chapter 6 we draw conclusions and propose future works based on this thesis.

2 STATE OF THE ART

In this Chapter, we discuss the relevant state-of-the-art considering the main contributions of this thesis: CUTiS (standing for ClUstering Trajectory Stream), CUTiS* and Net-CUTiS which is our proposed solution for clustering of road network constraint trajectory data. Specifically, this Chapter has the following structure. In Section 2.1, the works about trajectory data management. Then, we discuss about the trajectory pattern mining in Section 2.2.

It is worth noticing that the contributions of this thesis can be included in the area of trajectory data management which deals with the traveling history of a moving object. A more comprehensive survey on trajectory data management can be found in (DENG *et al.*, 2011). The works in this field started more than two decades ago, motivated by the need to store, index, model, and query trajectory data. We focus and present in details only the related work relevant to the contributions of this thesis

2.1 Trajectory Data Management

Mobility data has been fostered by the widespread diffusion of wireless technologies, such as the call detail records from mobile phones and the GPS tracks for navigation devices, society-wide proxies of human mobile activities. These data open new opportunities for discovering the hidden patterns and models that characterize the trajectories humans follow during their daily activity, network traffic flow analysis and travel time reliability analysis. The huge volume of collected trajectories opens new opportunities for discovering the hidden patterns about mobility behaviors. These patterns may apply to characterize the individual mobility as well as groups sharing similar trajectories for a certain time period.

A trajectory is usually defined as the spatio-temporal evolution of moving objects (e.g., persons, vehicles, animals). This evolution is typically represented as a sequence of positional observations represented by x and y coordinates of time-stamped sample points. Mining huge volumes of trajectories is very time-consuming, as we need to access different samples of the trajectories or different parts of a trajectory many times. This calls for effective data management techniques that can quickly retrieve the trajectories (or parts of a trajectory) needed. Different from a moving object database that is concerned with the current location of a moving object, the trajectory data management introduced in this section deals with the real displacement (trajectory shape) of a moving object.

2.1.1 Trajectory Index

Before designing an access method, we need to be aware of the nature of the data, the types of queries and the method's usefulness. Range queries are typical queries which retrieve the trajectories intersecting a spatial (or spatio-temporal) range. A trajectory can be treated as spatial (3D) data itself, and thus may be supported by a spatial access method. By regarding the time as the third dimension besides the 2D geographical space, building an index 3D-Rtree (THEODORIDIS *et al.*, 1996) on a trajectory dataset, it is possible to answer such range queries by retrieving the nodes within the 3D query box. The 3D-Rtree works well for indexing trajectories generated in the near recent (e.g., in the past few hours). When the time span of the trajectories to be indexed lasts for a long period (i.e., more segments of newly generated trajectories will be inserted into a 3D-Rtree index), the overlap among 3D boxes bounding segments of different trajectories occur more often (ZHENG, 2015).

Another problem not addressed by any of the traditional index for spatio-temporal data (Quadtree (TZOURAMANIS *et al.*, 1998), R-tree and variations (NASCIMENTO *et al.*, 1999)) is the preservation of trajectories. The first proposal in this direction was Spatio-Temporal R-tree (called STR-tree) which organizes line segments not only according to spatial properties, but also by attempting to group the segments according to the trajectories they belong to. STR-tree is an extension of 3D R-tree however STR-tree tries to keep line segments belonging to the same trajectory together. As a result, when intersecting a new line segment, the goal is to insert it as close as possible to its predecessor in the same trajectory. The second, the Trajectory-Bundle tree (hereafter called TB-tree) which aims only for trajectory preservation and leaves other spatial properties aside. Its main idea is to bundle segments from the same trajectory into the leaf nodes of the R-tree. The trajectory is distributed over a set of disconnected leaf nodes. Both access methods were proposed in (PFOSER *et al.*, 2000).

(POPA *et al.*, 2010) proposed PARINET an index to efficiently retrieve the trajectories of objects moving on networks. The structure of PARINET is based on a combination of graph partitioning and a set of composite B_+-tree local indexes. PARINET is designed for historical data and relies on the distribution of the data over the network as for historical data, the data distribution is known in advance. Because the network can be modeled using graphs, the partitioning of the trajectory data is based on graph partitioning theory and can be tuned for a given query load. The data in each partition is indexed on the time component using B_+-trees. (POPA *et al.*, 2011) proposed T-PARINET which is an extension of PARINET for trajectory data

flows.

2.1.2 Distance and Similarities between Trajectories

When answering clustering trajectories, we need to calculate the distance (alternatively, similarity) between two trajectories. The distance discussed below is divided by for Euclidean Space and Road Network. Although there is significant research work performed on the trajectory similarity on moving objects, the vast majority of the proposed approaches assumes that objects can move freely without any motion constraints.

The work of (ANKERST *et al.*, 1999) presents a time and space distance function in Euclidean space. From a conceptual viewpoint to compute the distance we need to compute the infinite set of distances for each timestamp (e.g., and, afterward, to aggregate them by an integral). The paper (FRENTZOS *et al.*, 2007b) proved that this integral can be computed on $O(1)$ and can be efficiently computed by Trapezoid Rule. (FRENTZOS *et al.*, 2007b) provided also bounds for the approximation error. This distance function is well explained in Chapter 4.

Some outlier points from a trajectory may cause a big distance between trajectories, the concept of the Longest Common Subsequence (LCSS) (VLACHOS *et al.*, 2002) is employed to address this issue. The LCSS-based distance allows one to skip some noise points when calculating the distance of trajectories, using a threshold δ to control how far in time we can go in order to match one point from a trajectory to a point in another trajectory. Another threshold ε is used to determine whether two points (from two different trajectories) are matched. The distance function Edit Distance on Real Sequence (EDR) (CHEN *et al.*, 2005) is similar to LCSS in using a threshold ε to determine a match, while assigning penalties to the gaps between two matched subtrajectories.

The work of (LEE *et al.*, 2007) proposed a distance function $distance(ST_{i,k}, ST_{j,k})$ for any (sub) trajectories (let's say, $ST_{i,k}$ and $ST_{j,k}$) which is a weighted sum of three terms: (1) The aggregate perpendicular distance ($d_{\perp}$) that measures the separation between two trajectories, (2) the aggregate parallel distance ($d_{\parallel}$) that captures the difference in length between two trajectories, and (3) the angular distance (d_{θ}) that reflects the orientation difference between two trajectories. Formally,

$$distance(ST_{i,k}, ST_{j,k}) = \omega_1 * dist_{\perp}(ST_{i,k}, ST_{j,k}) + \omega_2 * dist_{\parallel}(ST_{i,k}, ST_{j,k}) + \omega_3 * dist_{\theta}(ST_{i,k}, ST_{j,k})$$

(2.1)

These components were adapted from similarity measures used in the area of pattern recognition (CHEN *et al.*, 2003). Each sub-trajectory is composed of line segments, imagine that L_i belongs to $ST_{i,k}$ and L_j belongs to $ST_{j,k}$. Then, $L_i = s_i e_i$ and $L_j = s_j e_j$. Here, s_i, e_i, s_j, and e_j represent d-dimensional points. The three components which compose the distance function are illustrated in Figure 1. The angular distance component is used in this thesis, and more details will be given in Chapter 4.

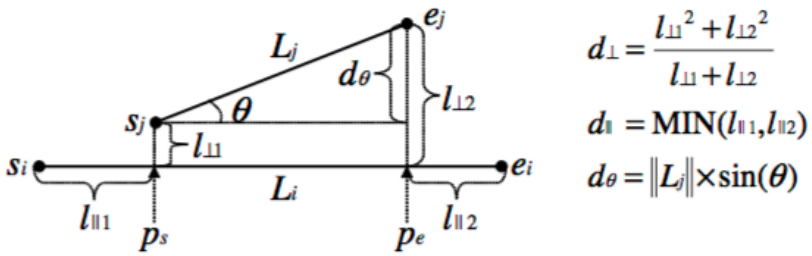


Figura 1 – Three components of the distance function for line segments from(LEE *et al.*, 2007).

The work of (NANNI; PEDRESCHI, 2006) proposes a similarity measure of trajectories of objects along time and therefore is computed by analyzing the way the distance between the objects varies. More precisely, it restricts to consider only pairs of contemporary instantiation of objects, i.e., for each time instant the similarity compares the positions of the objects at that moment, thus aggregating the set of distance values obtained this way. The distance between two trajectories T_j and T_k is computed as follows

$$D(T_j, T_k)|T = \frac{\int_T d(T_j(t), T_k(t))\,\mathrm{d}t}{|T|} \tag{2.2}$$

where $d(T_j(t), T_k(t))$ is the Euclidean distance, T is the temporal interval over which trajectories $T_j(t)$ and T_k exist, and $T_i(t)(i \in \{1,2\})$ is the position of object T_i at time t. Such a definition requires a temporal domain common to all objects, which, in general, is not a hard requirement (NANNI; PEDRESCHI, 2006).

The first work to propose a trajectory similarity function on networks was (HWANG *et al.*, 2005). The authors propose a simple similarity measure based on POIs (points of interest). They propose a filtering method based on spatial similarity and refining similar trajectories based on temporal distance. In order to determine the spatial similarity between trajectories, they define

that two trajectories are similar in space by a set of pre-defined points of interest P if all points of P lie in both trajectories, otherwise they define the two trajectories as dissimilar.

Other papers as (TIAKAS *et al.*, 2006; TIAKAS *et al.*, 2009) also proposed spatio-temporal similarity function for trajectory on the road network. There are three different methods proposed in order to retrieve similar trajectories in space-time: (1) searching for similar trajectories based on spatio-temporal distance between trajectories; (2) filtering trajectories based on temporal similarity and refining similar trajectories based on spatial distance; (3) filtering trajectories based on spatial similarity and refining similar trajectories based on temporal distance.

Recently (PELEKIS *et al.*, 2016) proposed a new distance function for two timelines. A mobility timeline is a sequence of LifeSteps (firstly defined (PELEKIS *et al.*, 2014; PELEKIS *et al.*, 2015)), and each LifeStep can be abstracted as a pair of values (θ, κ), where θ is a spatio-temporal value that provides an approximation of a portion of the movement of the user (i.e., an Minimum Bounding Box (MBB)), and κ provides a corresponding text description (i.e., as a set of keywords) given semantics to θ. The distance metric proposed can be decomposed into two parts, one measuring the distance of the spatio-temporal components and another measuring the distance of the textual components. (PELEKIS *et al.*, 2016) applies this distance function in the proposed clustering algorithm Sem-T-Optics which follows T-Optics (NANNI; PEDRESCHI, 2006) clustering algorithm.

2.1.3 Trajectory Compression

Trajectory Compression aims to compress the size (e.g., the cardinality of the set of time-stamped positions) of trajectory, but it does not affect its usefulness. By compressing trajectories, we can have an overhead reduced in communication, processing and data storage. There are two categories for trajectory compression: offline compression, which reduces the size of a trajectory already generated. Another one is online compression, which reduces the size of a trajectory while the object moves.

Douglas Peucker (DOUGLAS; PEUCKER, 1973) is the most effective batch algorithm for compression (POPA *et al.*, 2015). It takes as input a data series of 2D points and a user-defined threshold value to output subseries of the input data. The idea of Douglas Peucker is to replace the original trajectory by an approximate line segment. It recursively partitions the original trajectory into two sub-trajectories by selecting the point in the original trajectory which

contributes to the biggest error (e.g., perpendicular Euclidean distance). We call this point as splitting point. This process continues until the error between the approximation and the original trajectory is below a specified error.

Douglas Peucker is a commonly used compression technique, however, it is not suitable for moving object trajectory applications. Mostly because it operates on the basis of perpendicular distance. Due to the continuous nature of moving objects, the proposals of (MERATNIA; ROLF, 2004) are related to Douglas Peucker, but taking both spatial and temporal factors into account.

The paper of (KELLARIS *et al.*, 2013) proposes two trajectory compression techniques, one online and another offline. As the first step, it applies a map matching algorithm to associate each trajectory position to one road segment on the road network. As a second step, it compares if a sub-trajectory can be represented by the shortest path between its endpoints. The authors adopted the Minimum Description Length (MDL) (GRÜNWALD *et al.*, 2005) principle to compress, in this way a sub-trajectory is replaced by the shortest path in the original trajectory when this shortest path has the least value for MDL.

(POTAMIAS *et al.*, 2006) proposed two algorithms for online trajectory compression. The algorithm uses the coordinates, speed, and orientation of the current point in order to calculate a safe area where the next point might be located. If the next incoming points belong to the safe area, it can be ignored. In order to define a safe area, basically it can be calculated using the last point or the last chosen point.

The paper (POPA *et al.*, 2015) compresses trajectories which are road network constraint. As a first step, it discovers the network paths (a sequence of consecutive road segments) or which its road segments are all crossed by a set of the same trajectories. The second step is to describe all trajectories in the data set using these paths (mapping function). The third step is to compress all trajectories by applying the Douglas Peuker algorithm using as a function the Synchronous Network Distance. In Chapter 5, we will explain our proposed distance function for the trajectories constraint to road network movement and it uses (POPA *et al.*, 2015) compression algorithm.

2.2 Trajectory Pattern Mining

In this section, we overview two major categories of moving together patterns (e.g., moving object clusters and trajectory clustering) that can be discovered from a single trajectory

or a group of trajectories.

2.2.1 Moving Together Patterns

Clustering algorithms, which group similar data to reveal overall distribution patterns and interesting correlations in datasets, has achieved brilliant success in analyzing static datasets (ESTER *et al.*, 1996; ANKERST *et al.*, 1999; MACQUEEN, 1967; GUHA *et al.*, 1998; ZHANG *et al.*, 1996; ESTER *et al.*, 1998; YIU; MAMOULIS, 2004; DAI; LIN, 2012; HE *et al.*, 2011; SILVA *et al.*, 2014a). These works proposed algorithms for three different clustering paradigms, i.e., k-partitioning, density-based and hierarchical clustering. However, all these solutions are clustering algorithm for static datasets. The work of (ESTER *et al.*, 1998) proposes an approach to generate clusters in an incremental way. An extensive research has also been done for clustering moving objects, which could catch interesting pattern changes during the motion process and provide better insight into the essence of the mobile data points.

(LI *et al.*, 2004) applies micro-clustering of moving objects, thus obtaining algorithms that dynamically maintain bounding boxes of clusters. A micro-cluster denotes a group of objects that are not only close to each other at the current time, but also likely move together for a while. Due to the difference of positions and velocities of objects within a micro-cluster, the objects tend to scatter after a period of time, which may break the requirement of micro-cluster since they will not be adjacent to each other any more. However, the numbers of maintenance events (e.g., *split* and *collision* of micro-cluster(s)) involved are usually prohibitively large.

The paper (JENSEN *et al.*, 2007) proposes a fast and effective scheme for the continuous clustering of moving objects. It presents a dynamic summary data structure for clusters that enables frequent updates to the data without the need for global reclustering. An average-radius function is used that automatically detects a cluster split events, which, in comparison to (LI *et al.*, 2004), eliminates the need to maintain bounding boxes of clusters with large amounts of associated violation events. Moreover, (JENSEN *et al.*, 2007) considers in its strategy that each cluster has a maximum capacity (e.g., radius which means the furthest distance between two objects that belong to the cluster), which might not make sense for density-based clusters.

(LI *et al.*, 2013) proposes a framework that efficiently supports the online discovery of moving objects that travel together. The framework adopts a sampling-independent approach, monitors clusters continuously and records the histories of clusters. The evaluation of groups is

only dependent on the most recent history, so that the framework is amenable to online processing. The work of (LI *et al.*, 2013) tracks all cluster evolution patterns and define the conditions that may lead to these evolutions.

Another related work is (LEE *et al.*, 2014). It focuses on the cluster evolution tracking problem on highly dynamic networks. (LEE *et al.*, 2014) takes the event evolution tracking task in social streams as an application, where a social stream and an event are modeled as a dynamic post network and a dynamic cluster respectively. (LEE *et al.*, 2014) summarizes the network by a skeletal graph and monitor the updates to the post network by means of a sliding time window. It formalizes cluster evolution patterns using a group of primitive evolution operations and their algebra. Two incremental computation algorithms (split and merge clusters) are developed to maintain clusters and track evolution patterns as time rolls on and the network evolves. Unlike previous approaches, their evolution tracking algorithm performs incremental bulk updates in real time. (LEE *et al.*, 2014) is related to our work, considering that all cluster evolution patterns are observed. However,it works for social networking, moreover, it does not consider the update operation as we consider that a moving object trajectory can change its position as time goes by.

A very related topic to this study is to discover collective patterns among moving objects, as flock (GUDMUNDSSON; KREVELD, 2006; GUDMUNDSSON *et al.*, 2004), swarm (LI *et al.*, 2010a), convoy (JEUNG *et al.*, 2008a; JEUNG *et al.*, 2008b), travelling companions (TANG *et al.*, 2012; TANG *et al.*, 2013), herds (HUANG *et al.*, 2008), gathering (ZHENG *et al.*, 2013), among others (ZHENG, 2015). In the following, we discuss these patterns in more details.

These patterns can be differentiated between each other based on the following factors: the shape of a group, the number of objects in a group, and the duration of a pattern. Specifically, a flock is a group of objects that travel together within a disk of any user-specified size for at least k consecutive timestamps. A major concern with flock is the predefined circular shape, which may not well describe the shape of a group in reality, and therefore may result in the so-called lossy-flock problem. To avoid rigid restrictions on the size and shape of a moving group, the convoy is proposed to capture the generic trajectory pattern of any shape by employing the density-based clustering. Instead of using a disk, a convoy requires a group of objects to be density connected during k consecutive time points. While both flock and convoy have a strict requirement for consecutive time period, swarm is a cluster of objects lasting for at least k

(possibly non-consecutive) timestamps. While convoy and swarm patterns need to load entire trajectories into memory for a pattern mining, the traveling companion uses a data structure (called traveling buddy) to continuously find a convoy/swarm like patterns from trajectories that are being streamed into a system. So, the traveling companion patterns can be regarded as an online (and incremental) detection fashion of convoy and swarm (ZHENG, 2015). As an online algorithm to find flocks, the work of (VIEIRA *et al.*, 2009) finds online flocks with the longest duration in moving object data stream.

(HUANG *et al.*, 2008) proposes new conceptual definitions for a spatio-temporal pattern named herd and four types of herd evolvements: expand, join, shrink, and leave. We will discuss all these evolution patterns in the next section. Herd evolvements are identified through measurements of Precision, Recall, and F-score.

Finally, (ZHENG *et al.*, 2013) discovers patterns, called gathering, which is a trajectory pattern modelling various group incidents such as celebrations, parades, protests, traffic jams and so on. Formally, each cluster of a gathering should contain at least m_p so-called participators, which refer to the objects appearing in at least k_p groups of this gathering. A key observation is that these incidents typically involve large congregations of individuals, which form durable and stable areas with high density. Since the process of discovering gathering patterns over large-scale trajectory databases can be quite lengthy, (ZHENG *et al.*, 2013) further develop a set of well thought out techniques to improve the performance. These techniques, including effective indexing structures, fast pattern detection algorithms implemented with bit vectors, and incremental algorithms for handling new trajectory arrivals.

The published approaches to find these patterns are very sensitive to specific parameters and find these patterns offline for historical datasets.

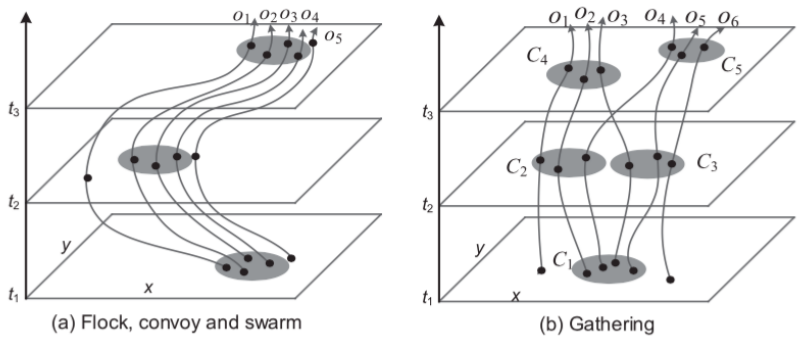


Figura 2 – Example of Flock, Convoy, Swarm and Gathering from (ZHENG, 2015)

Figure 2 illustrates the majority of these patterns. Imagine the parameter k=2, a

group (o_2, o_3, o_4) is a flock from t_1 to t_3 (Figure 2 (a)). A convoy can be composed by including o_5 into the group, since (o_2, o_3, o_4, o_5) is density-based connected from t_1 to t_3. A swarm is derived from the five objects during the non-consecutive time period t_1 and t_3. In the Figure 2(b), if we consider $k_p = 2$ and $m_p = 3$, then (C_1, C_2, C_4) is a gathering. As C_5 is too far away from C_2 and C_3, (C_1, C_3, C_5) is not a gathering.

Another paper in this context is (LEE *et al.*, 2015) that proposes a framework of mining historical trajectory dataset, in order to find UT-patterns which are informally defined as sets of moving objects that are closely related in terms of location, time, or both.

There exist some approaches for clustering of moving objects data stream. The majority of data-stream clustering algorithms are based on k-means (SILVA *et al.*, 2013). These clustering algorithms are incompetent to find clusters of arbitrary shapes and cannot handle outliers. There are two main algorithms for data stream clustering density-based: DenStream (CAO *et al.*, 2006) and D-Stream (CHEN; TU, 2007). In general, data stream clustering algorithm develops suitable data structures for storing statistic summaries about clusters and save the computation for incremental clustering from one snapshot data to the next (as a feature vector (DenStream (CAO *et al.*, 2006)) or grid data structure (D-Stream (CHEN; TU, 2007)) which are extensively used in many data stream clustering algorithms (SILVA *et al.*, 2013)).

It is unfeasible to use these data structures, since our problem (1) deals with moving object sub-trajectory data; (2) does not consider fading time window or decay factor as in D-Stream (CHEN; TU, 2007), since we aim at tracking moving objects, the process to remove a moving object from the dataset occurs when a moving object stops to send its positions for a long time (until some threshold is reached). Furthermore, we consider in our problem that all the moving objects change their sub-trajectories every time window different from DenStream (CAO *et al.*, 2006) and we can receive new moving object data; (3) we aim at capturing the similarity in time, space and angular movement of moving objects sub-trajectories during a time window.

The papers (GUHA *et al.*, 2000; AGGARWAL *et al.*, 2003) proposed clustering approaches for data streams. Different from the previous, they do discover density-based clusters. However, they are not addressed for trajectory data.

The next section discusses how groups of moving objects, which shares the same space at the same time, evolve as time goes by. We call this phenomenon as cluster evolution patterns.

2.2.2 *Cluster Evolution Pattern*

Imagine the following scenario, illustrated in Figure 3: at time t, there are three clusters and some outliers in the data set. Consider that each point in the data set is a moving object and each cluster represents a high density area, for example. After δt units of time, the objects of clusters C_1 and C_2 moved into the same space and they are merged to only one cluster. Observe that on Figure 3, we label to cluster C_1 the objects that belonged to cluster C_2 on the previous time. This phenomenon is called cluster evolution pattern and it can happen in a real scenario (imagine a highway that has two roads which merge at some point). Note that, for cluster C_3, more moving objects were added to the dataset between $t+\delta t$ and $t+2\delta t$. Cluster C_3 at time $t+2\delta t$ can represent, for example, the increase of vehicles in an area and it may represent a congested area.

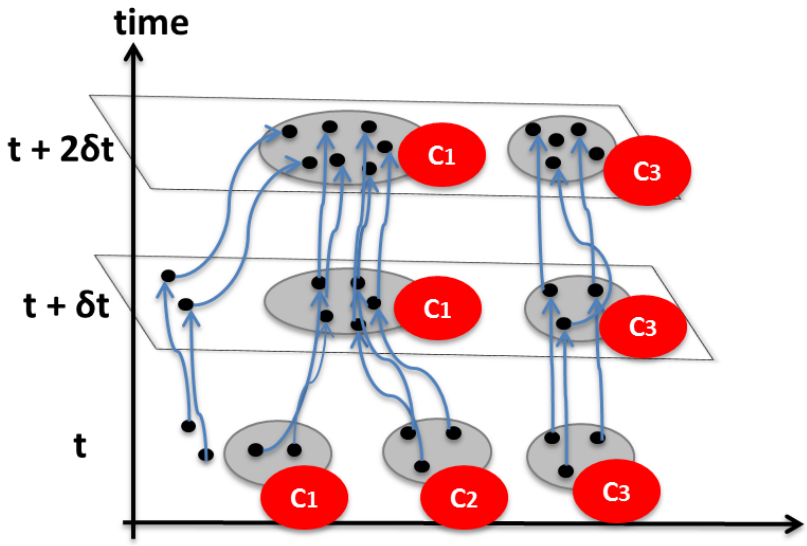


Figura 3 – Example of clustering moving objects at three consecutive timestamps

The cluster evolution of moving objects should take into account the objects' previous movements, their updated movement as time goes by, including the deletion and insertion of other moving objects. As we shall see, doing so enables us to capture each clustering change as it occurs during the continuous motion process. Thus, we aim at tracking the cluster evolution patterns at each moment from such dynamic dataset. Typical cluster evolution patterns include appear, disappear, expand, shrink, split, merge and survive (SPILIOPOULOU *et al.*, 2006).

Our paper (SILVA *et al.*, 2014b) presents a naive approach to detect density-based cluster evolution patterns, considering that the clustering algorithm is executed over all dataset.

It is worth noting that this approach is not optimal since we compute clusters at every δt units of time and for each execution we will evaluate the cluster evolutions. Motivated by that, we proposed the framework CUTiS, which is able to automatically detect the cluster's evolutions without needing to recluster the data from time to time. CUTiS is the major contribution of this thesis.

2.2.3 Trajectory Clustering

Usually, trajectory analysis is done off-line, i.e., by applying data analysis and mining techniques on the previously collected data (ZHENG, 2015). This allows characterizing the past movements of the objects but not the current mobility patterns. Nowadays, many services exist that involve moving objects to report their trajectory continuously (e.g., every second or every minute). Analyzing these data in real time may bring a real added-value in the comprehension of the city dynamics, and the detection of regularities as well as an anomaly, which is essential for decision-making. Among these patterns, in this thesis, we consider the trajectory group constitution and evolution, based on sub-trajectory cluster analysis in real time. Such discovery may help the search for effective re-engineering of traffic, or dynamically detecting events or incidents, e.g., at a city level.

In the next section, we discuss clustering trajectory data which finds patterns according to the real displacement of moving objects in Euclidean Space.

2.2.3.1 Clustering of Trajectory Data in Euclidean Space

Trajectory clustering is a topic that has gained great interest in the literature (NANNI; PEDRESCHI, 2006; PELEKIS *et al.*, 2011; LEE *et al.*, 2007; LI *et al.*, 2010b; YU *et al.*, 2013; MAO *et al.*, 2016). The authors in (NANNI; PEDRESCHI, 2006) proposes a time-focused clustering of trajectories based on OPTICS algorithm (ANKERST *et al.*, 1999) and present a time and space distance function in Euclidean space. The paper (PELEKIS *et al.*, 2011) studied how to cluster uncertain trajectories. However, these approaches tend to generate clusters for the entire trajectory dataset, hence the fine-grained spatio-temporal relationships between moving objects are lost. The paper (LEE *et al.*, 2007) proposes TraClus which is an approach to discover density based sub-trajectory clusters based on DBSCAN algorithm (ESTER *et al.*, 1996). For trajectory comparison, TraClus focused on spatial criteria and ignored the time dimension. In this thesis, we compared our approach with TraClus.

Some of the applications may like daily patterns or every minute patterns if they are online. For example, a private company's transportation that tags and tracks its vehicles as they move around. The devices report the trajectories on a continual basis as streams (e.g., every minute, every hour). With such data, it is possible to support drivers' decisions related to displacement through the city in real time (i.e., traffic jam or movement habits through the city can be found by trajectory cluster analysis). Such discovery may help the search for effective re-engineering traffic solutions in the context of smart cities.

There exist some approaches for clustering trajectories data stream. The work (LI *et al.*, 2010b) proposed efficient algorithms for maintaining and updating the clusters when new trajectories stream are received. However, it does not consider the temporal aspects of the trajectories. As such, moving objects whose trajectories are in the same cluster may not actually stay together temporally. Paper (YU *et al.*, 2013) proposed a trajectory clustering algorithm to extract some patterns similar to the convoy pattern (JEUNG *et al.*, 2008c). The paper (MAO *et al.*, 2016) proposes an online trajectory clustering over a sliding window composed of two components: a micro-clustering that extracts the summary of trajectory stream in the window, and a macro-clustering component that re-clusters the previously extracted summaries according to user's request. However, both papers (YU *et al.*, 2013; MAO *et al.*, 2016) do not track the cluster/group evolution neither the up-to-date moving objects trajectories as time goes by.

2.2.3.2 Clustering of Trajectory Data on Road Network

Note that the clustering methods discussed in the last section are dedicated for trajectories in free spaces (i.e., without a road network constraint). In this section, we will discuss trajectory clustering approaches which only consider road network constraint movement.

The work of (LI *et al.*, 2007) proposes a density-based algorithm called FlowScan, which finds the "hot routes" based on the definition of 'traffic density-reachable'. It is essentially a trajectory clustering algorithm based on traffic density, which clusters trajectories by dense line segment. The paper (HAN *et al.*, 2012) proposes NEAT a road network aware approach for fast and effective clustering of spatial trajectories of mobile objects travelling on road networks. This method takes into account the physical constraints of the road network, the network proximity and the traffic flows among consecutive road segments to organize trajectories into spatial clusters. The clusters discovered by NEAT are groups of sub-trajectories which describe both dense and highly continuous traffic flows of mobile objects.

(ROH; HWANG, 2010) proposed a new distance function for trajectory based on Hausdorff distance which measures how far two trajectories of a metric space are from each other. It considers the network constraints to derive the proposed similarity measure with a focus on shortest path distance. Moreover, it proposes a trajectory clustering algorithm based on hierarchical clustering. It is not an online algorithm neither considers the time as a metric to measure the similarities between trajectories as we do.

The paper (KHARRAT *et al.*, 2008) proposes NETSCAN which finds density based clusters. The minimal required density for a road segment is α moving objects crossing it. At first, NETSCAN finds the dense paths which mean a sequence of neighbor road segments. The maximal density difference between neighbour segments is equal to ε to be in the same dense path. NETSCAN does not ensure the same moving objects cross a dense path. The trajectory clustering algorithm consists in grouping the trajectories according to their similarity to each dense path generated. For each dense path, it computes the similarity with each trajectory of the dataset. If the similarity is above the threshold value σ, then the trajectory is kept in the cluster. More precisely, the common part between the trajectory and the dense path is added to the cluster. The number of returning clusters is equal to the number of dense paths. The time is relaxed and the same moving object trajectory can be associated with different clusters. NETSCAN is compared with the evaluation of our approach for clustering road network constraint trajectories.

From the best of authors' knowledge, there is no incremental approach for trajectory clustering online on the road network.

In the next chapter, we discuss how to discover frequent evolution patterns on moving object database.

3 FREQUENT EVOLUTION PATTERN DISCOVERY

The problem addressed in this chapter corresponds to the **Challenge 3** presented in the Section 1.2.

Typical cluster evolution patterns include appear, disappear, expand, shrink, split, merge and survive (SPILIOPOULOU *et al.*, 2006). The cluster evolution of moving objects should take into account objects previous movements, their updated movement as time goes by, including the deletion and insertion of other moving objects. As we shall see, doing so enables us to capture each clustering change as it occurs during the continuous motion process. In this chapter, we aim at discovering and detecting frequent cluster evolution patterns on a dynamic moving object dataset.

Many successful and scalable approaches have been proposed (ESTER *et al.*, 1996; MACQUEEN, 1967; ESTER *et al.*, 1998; YIU; MAMOULIS, 2004; DAI; LIN, 2012; SILVA *et al.*, 2014a), which have achieved success on clustering static mobility datasets. As we mentioned before, they answer the question "Where are the moving object clusters?". However, they can not answer advanced queries like "How moving object clusters evolve over the geographical space?". This type of query is solved using the approaches proposed on (XU *et al.*, 2014; JENSEN *et al.*, 2007; LI *et al.*, 2013).

In this chapter, we do not only detect the cluster evolution patterns, but also address queries like "Which are the frequent patterns of cluster evolution of moving objects trajectories?". For example, on applications which aim at analysing the traffic behavior. The existing approaches to detect the evolution patterns can discover where the congested areas are within the city at each moment and how they evolve in time. However, they can not detect if the congested areas are frequent throughout the day, the week or during the month.

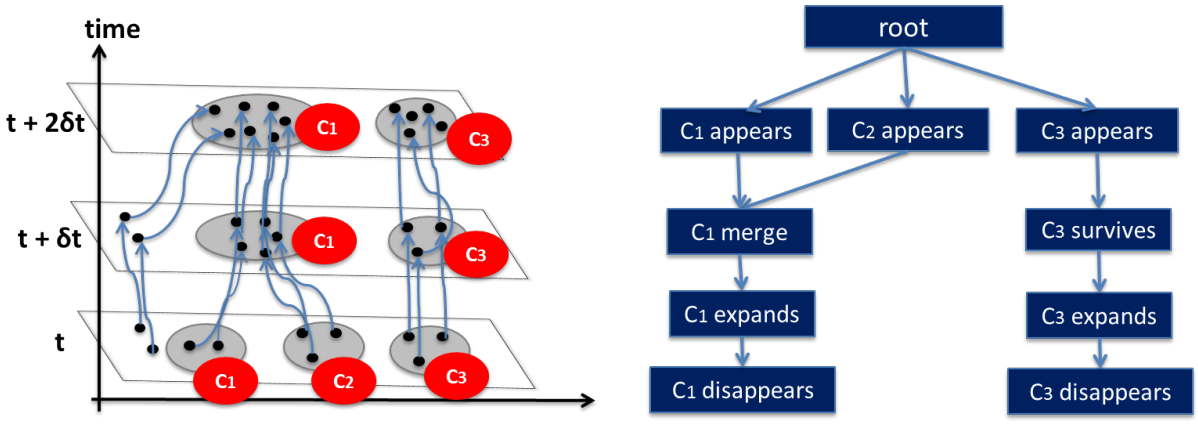
Many applications need to cope with cluster changes in the mobility domain, among them traffic management, delivery logistics and crowd monitoring. It is also necessary to provide insights about the nature of cluster change: Is a cluster corresponding to a group of vehicles simply disappearing or are its members migrating to other clusters? Is a new emerging cluster reflecting a new target group of vehicles or does it rather consist of existing customers whose preferences shift? (SPILIOPOULOU *et al.*, 2006).

We focus on modeling cluster evolution patterns, on detecting the evolutions over time and the frequent sequence evolution patterns on a dynamic moving object dataset. It is computationally costly to calculate the frequency for every possible sequence of cluster evolution

in the dataset, we aim to avoid this exhaustive computation.

Our proposed approach here is key for mobility applications, for instance, applications that need to discover mobility pattern on moving object's data. A mobility pattern represents the common behavior of a group or subgroup of trajectories, obtained as a result of a data mining algorithm. For example, to find gathering pattern on trajectory data (ZHENG *et al.*, 2013). Informally, a gathering represents a group event or incident that involves the congregation of objects (e.g., vehicles, people, animals). Examples of gatherings may include celebrations, parades, large-scale business promotions, protests and traffic jam.

For instance, we may describe a traffic jam as a sequence of evolution patterns as we show on Figure 4(a): two clusters C_1 and C_2 appear at time t. After δt units of time, C_1 and C_2 move closer to each other and they merge into one cluster, we label it as C_1. At $t + 2\delta t$, imagine that C_1 expands if the traffic jam gets worse. Or C_1 shrinks, otherwise. When the traffic jam is over, the cluster evolution of C_1 is disappearing. If this sequence $(appear, merge, expand|shrink, disappear)$ occurs frequently in the dataset, we claim that the traffic jam is a frequent pattern. Therefore, to solve our problem in this work, we model all cluster evolution on a Graph Evolution Cluster over Time (Δ_{evol}) as show in Figure 4(b). Then, we map our problem on discovering frequent paths on Δ_{evol}. The frequent graph paths are frequent sequences of evolution patterns that occurs in the dataset. We will discuss our solution on Section 3.2.



(a) Data records at three timestamps (b) Graph Evolution Cluster over Time

Figura 4 – Example of detecting and discovering frequent cluster evolution pattern

On Section 3.2 we present our approach, which does not require to check all the combinations of possible evolutions to discover which are the most frequent. Furthermore, we

present a naive approach to detect clusters evolution patterns, considering that the clustering algorithm is executed over all datasets. It is worth noting that this approach is not optimal since we will compute clusters at every δt units of time and for each execution we will evaluate the cluster evolutions. After, we discover the frequent sequences of cluster evolution. Chapters 4 and 5 propose an incremental clustering algorithm which is able to online detect cluster evolution. Our incremental clustering algorithm is an improvement of this first proposal.

3.1 Definitions

In this section, we start by defining the basic notions of moving object, moving object position and consistent set of moving object positions which are important for the explanation over this chapter.

Definition 3.1.1. (Moving object) A *moving object* is a pair $\omega = (oid, \rho)$ such that *oid* is a unique identifier for ω and $\rho(t) : \Re \rightarrow \Re^2$ is the trajectory of ω such that $\rho(t) = (lat, lon)$ maps each point in time $t \in \Re$ into a pair $(lat, lon) \in \Re^2$.

Definition 3.1.2. (Moving object position) Let $\omega = (oid, \rho)$ be a moving object and $t \in \Re$. The *position* of ω at time t is the triple $o = (oid, \rho(t), t)$.

Definition 3.1.3. (Consistent set of moving object positions at the same timestamp) Let Ω be a set of moving objects. Let O be a set of positions of moving objects in Ω. Then, O is *consistent* iff, for any $(i, p, t), (j, q, u) \in O$, $i \neq j$ and $t = u$.

We now focus on the evolution of a set of moving object positions that results from changes in the objects positions and from the deletion and insertion of moving objects in the dataset. These changes are modelled as events that represent the execution of one of the operations in the next definitions.

In what follows, let Ω be a set of moving objects and O^{Ω} be the set of all consistent set of moving object positions of moving objects in Ω.

Definition 3.1.4. (Insertion operation - there is a new moving object position) The *insertion operation* $o^+ : \Re \times \Omega \times O^{\Omega} \rightarrow O^{\Omega}$ is such that, for each time $t \in \Re$, each moving object $\omega = (oid, rho) \in \Omega$ and each set O of positions of objects in Ω at time t, $o^+(t, \omega, O) = P$ iff $P = O \cup \{(oid, \rho(t), t)\}$ (if O already contains the position of ω at t, then O remains unchanged).

Definition 3.1.5. (Deletion operation - there exists a deletion of a moving object position) The *deletion operation* $o^- : \Re \times \Omega \times O^\Omega \rightarrow O^\Omega$ is such that, for each time $t \in \Re$, each moving object $\omega = (oid, rho) \in \Omega$ and each set O of positions of objects in Ω at time t, $o^-(t, \omega, O) = P$ iff $P = O - \{(oid, \rho(t), t)\}$ (if O does not contain the position of ω at t, then O remains unchanged).

Definition 3.1.6. (Update operation - there exists an update on a moving object position) The *update operation* $o^* : \Re \times O^\Omega \rightarrow O^\Omega$ is such that, for each time $t \in \Re$ and each set O of positions of objects in Ω at time t, $o^*(t, O) = P$ iff $P = \{(oid, \rho(t), t)/(\exists (i, \rho) \in \Omega)(\exists u \in \Re)((i, \rho(u), u) \in O \wedge oid = i\}$.

Definition 3.1.7. (Event) An event is an instance to an insertion, deletion or update operation with a specific set of inputs.

To discover cluster evolution patterns, we first need to apply a clustering algorithm to our dataset. A spatial cluster is a group of entities in spatial proximity. Various clustering algorithms can be used, including partitioning based, hierarchical, model based, and density based. We already argued that a density-based clustering is more convenient for our work.

We first define the Jaccard similarity between two sets of consistent moving object positions at the same timestamp, which will be used in Algorithm 1 on Section 3.2.2 to detect the evolution patterns of clusters which contain sets of moving objects.

Given $O \in O^\Omega$, let O_{obj} denote the set of moving objects with positions in O:

$$O_{obj} = \{oid/(\exists (oid, \rho) \in \Omega)(\exists t \in \Re)((oid, \rho(t), t) \in O\}$$

Definition 3.1.8. (Jaccard Similarity) Let $O, P \in O^\Omega$. The Jaccard Similarity of O and P is defined as:

$$Similar(O, P) = \frac{\|O_{obj} \cap P_{obj}\|}{\|O_{obj} \cup P_{obj}\|}$$

We use Jaccard Similarity to compare two clusters at two different timestamps and then, find out the clusters evolution as we will present soon. The following definition introduces the notion of cluster of moving object positions, omitting the details of the clustering method.

Definition 3.1.9. (Clustering of Moving Object Positions) Let O be a consistent set of moving object positions. A clustering of O is a partition $C_t = \{S_0, ..., S_m, Out\}$ of O where Out is called

the set of outliers. C_t is the set of clusters at time t and $\forall i \in \{0,...,m\}, S_i$ has the following form (sid_i, O_{S_i}) where sid_i is a unique identity and $O_{S_i} \subseteq O$ is a set of moving objects positions that belong to S_i.

In what follows, let Ω be a set of moving objects and C^{Ω} be the set of all possible clusterings of consistent sets of moving object positions of moving objects in Ω. For convenience, we assume that the empty set $\emptyset$ is in C^{Ω}.

Let $EP = \{expands, shrinks, survives, merges, disappears, appears, splits\}$ be a set of evolution modelled here as *actions* (see Section 3.2.1). In order to detect a frequent cluster evolution pattern, we model all evolutions on a Graph Evolution Cluster over Time presented in the following definition.

Definition 3.1.10. (Graph Evolution Cluster over Time) A graph evolution cluster over time over Ω is a labeled directed graph $\Delta_{evol} = (N, E, \Lambda)$ such that

- Δ_{evol} is acyclic and has a single source, called the root of Δ_{evol}
- Λ assigns a label of the form (p, S_i, t) to each node in N, except for the root, where $p \in EP$ and S_i is a cluster of a clustering of the set O of positions of moving objects in Ω at time $t \in \Re$; and Λ labels the root with empty set
- A node in N has an indegree greater than 1 iff it is labelled with 'merge'; otherwise the node has indegree equal to 1
- A node in N has an outdegree equal to 0 iff it is labelled with 'disappear'; otherwise the node has outdegree equal to 1

Figure 4 shows an example of a graph evolution cluster over time Δ_{evol}. Note that Δ_{evol} is almost a tree, except for the nodes whose labels contain $'merge'$. Indeed, the label (p, S_i, t) of a node indicates that cluster S_i was the result of applying action p to one or more clusters. Note that a cluster in a clustering appears, expands, shrinks, survives, merges or disappears into just one cluster; therefore, the outdegree of all nodes is equal to 1. Furthermore, a cluster in a clustering may be the result of a merge of other clusters; in this case, and only in this case, the indegree of the node is greater than 1.

In the following definitions, we define a subgraph and isomorphism of graphs. In order to calculate how frequent an evolution pattern is, we model each pattern as a subgraph H of Δ_{evol}. Then, we find all subgraphs of Δ_{evol} which are isomorphic to H. If the number of all such subgraphs is greater than a threshold σ, we consider that H is a frequent pattern.

Tabela 1 – Notation for this chapter

Notation	Meaning
S_i	a cluster
C_t	a set of clusters $\{S_0, ..., S_m, Out\}$ at time t
O	a set of consistent moving object positions
$Similar(O,P)$	Similarity function to detect cluster evolution
Δ_{evol}	Graph Evolution Cluster over Time
H_i	a set with all the frequent subgraphs on Δ_{evol} with cardinality equals to i
F	set of frequent subgraphs $\{H_1, ..., H_k\}$ in Δ_{evol}
σ	minimum support to detect frequent patterns

Definition 3.1.11. (Subgraph) Given the graph $G = \{V(G), E(G)\}$, where $V(G)$ and $E(G)$ are set of vertex and edge. A graph H=$\{V(H), E(H)\}$ is a subgraph if and only if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

Definition 3.1.12. (Isomorphism) An isomorphism of graphs G and H is a bijection f between the vertex sets $V(G)$ and $V(H)$. $f: V(G) \to V(H)$ such that any two vertices $u \in V(G)$ and $v \in V(G)$, such that $\{u, v\} \in E(G)$ if and only if $\{f(u), f(v)\} \in E(H)$.

Now, we are ready to define our problem statement.

Problem Statement Let O be a consistent set of moving object positions of moving objects that belongs to Ω at time t. Let C_t be a clustering of O. Assume that our dataset is dynamic, after δt units of time, O can change due to the operations previously defined in the Definition 3.1.4, 3.1.5 and 3.1.6, and let them be stored in $U_{t+\delta t}$. If $U_{t+\delta t} = \emptyset$, the set C_t does not change. Construct a procedure such that, given O, $U_{t+\delta t}$, C_t:

1. Discovers and detects the cluster evolution of each $S_i \in C_t$ at time $t + \delta t$
2. Generates Δ_{evol} over time as the Definition 3.1.10
3. Finds the frequent cluster evolution patterns $F = \{H_1, ..., H_k\}$ on Δ_{evol} with respect to support σ

The output contains all clusters $C_{t+\delta t}$. It also specifies $\forall S_i \in C_t$, S_i evolution pattern in $C_{t+\delta t}$ and a set of frequent evolution patterns $F = \{H_1, ..., H_k\}$, such that $\forall i \in \{1, ..., k\}, H_i$ is a set of paths with cardinality equals to i on Δ_{evol} that occurs more than a support σ.

Tabela 2 – Events that may result on Actions

Action \ Event	$o+$	$o-$	$o*$
$S_i \Rightarrow S'_j$	-	-	-
$\{S_{i_1},...,S_{i_k}\} \Rightarrow S'_j$	**X**	-	**X**
$S_j \Rightarrow \{S'_{i_1},...,S'_{i_k}\}$	-	**X**	**X**
$S_i \nearrow$	**X**	-	**X**
$S_i \searrow$	-	**X**	**X**
$S_i \Rightarrow \odot$	-	**X**	**X**
$\odot \Rightarrow S'_i$	**X**	-	**X**

3.2 Proposed Solution

Since we aim to propose a cluster evolution approach for moving object dataset, we need to monitor all cluster evolution patterns. With this idea in mind, we map our solution to ECA (Event Condition Action) rules. As already defined, an event is the execution of an operation. A condition represents all the predicates that must be met so that an event happens. If the condition is true, it results in an action that is a cluster transition from t to $t+\delta t$. These conditions are used on Algorithm 1 to find the cluster evolution patterns.

Table 1 describes the notation used throughout this chapter.

3.2.1 Definitions of Actions and Conditions

In this section, we define the actions, which represents cluster transitions (evolution) from t to $t+\delta t$. In each action/transition below, the enumerated sentences must be true. Assume that at time t the clustering of moving objects is $C_t = \{S_0,...,S_m,Out\}$. In time $t+\delta t$, after the operations in $U_{t+\delta t}$, $C_{t+\delta t} = \{S'_0,...,S'_m,Out'\}$. Let $\tau \in (0,1]$ (it is difficult to define this threshold, because $U_{t+\delta t}$ can have many operations that involve many moving objects. In our experiments, we prefer to use $\tau = 0,5$. This means two clusters of different time instant t and $t+\delta t$ should share at least 50% to be consider one was evolved to the other).

1. **Action**: $S_i \Rightarrow S'_j$

   (Cluster $S_i \in C_t$ *survives* as cluster $S'_j \in C_{t+\delta t}$).

   Condition: $Similar(O_{S'_j}, O_{S_i}) = 1$

2. **Action**: $\{S_{i_1},...,S_{i_k}\} \Rightarrow S'_j$

   (Two or more clusters $S_{i_1},...,S_{i_k} \in C_t$ are *merged* into a single cluster $S'_j \in C_{t+\delta t}$).

   Conditions:

a) $Similar(O_{S_{i_1}} \cup ... \cup O_{S_{i_k}}, O_{S'_j}) \geq \tau$

b) $\forall S_r \in C_t \setminus \{S_{i_1}, ..., S_{i_k}\}$,
$Similar(O_{S_{i_1}} \cup ... \cup O_{S_{i_k}} \cup O_{S_r}, O_{S'_j}) < \tau$

3. **Action**: $S_j \Rightarrow \{S'_{i_1}, ..., S'_{i_k}\}$

(Cluster $S_j \in C_t$ *splits* into two or more clusters $S'_{i_1}, ..., S'_{i_k} \in C_{t+\delta t}$).

Conditions:

a) $Similar(O_{S'_{i_1}} \cup ... \cup O_{S'_{i_k}}, O_{S_j}) \geq \tau$

b) $\forall O_{S'_r} \in C_{t+\delta t} \setminus \{O_{S'_{i_1}}, ..., O_{S'_{i_k}}\}$,
$Similar(O_{S'_{i_1}} \cup ... \cup O_{S'_{i_k}} \cup O_{S'_r}, O_{S_j}) < \tau$

4. **Action**: $S_i \nearrow$

(Cluster $S_i \in C_t$ *expands* w.r.t. its size).

Condition: $\exists S'_j \in C_{t+\delta t}$ such that $S_i \subset S'_j$ (that is, S_i is a strict subset of S'_j).

5. **Action**: $S_i \searrow$

(Cluster $S_i \in C_t$ *shrinks* w.r.t. its size).

Condition: $\exists S'_j \in C_{t+\delta t}$ such that $S'_j \subset S_i$ (that is, S'_j is a strict subset of S_i).

6. **Action**: $S_i \Rightarrow \odot$

(Cluster $S_i \in C_t$ disappears).

Condition: $\forall S'_j \in C_{t+\delta t}$, $Similar(O_{S_i}, O_{S'_j}) < \tau$

7. **Action**: $\odot \Rightarrow S'_i$

(Cluster $S'_i \in C_{t+\delta t}$ appears).

Condition: $\forall S_j \in C_t$, $Similar(O_{S_j}, O_{S'_i}) < \tau$

Table 2 shows the events that may result in each action explained above. We relate them using 'X' in the cells of Table 2. When an event may trigger an action, there are conditions that must be checked. These conditions are defined on each action above. In the next subsection, we will present a naive approach to detect clusters evolution patterns using the conditions above, considering that the clustering algorithm is executed over all datasets. It is worth noting that this approach is not optimal since we will compute clusters at every δt units of time and, for each execution, we will evaluate the cluster evolutions.

3.2.2 Naive Approach

Considering C_t as all the clusters found at time t and $C_{t+\delta t}$ as all clusters found at time $t + \delta t$, we propose a naive strategy (Algorithm 1) to detect the evolution from C_t to $C_{t+\delta t}$.

Algorithm 1: Find each cluster evolution pattern

Input: Similarity matrix M, $C_t, C_{t+\delta t}, \tau, \Delta_{evol}$

begin
  if $\Delta_{evol} = \emptyset$ **then**
    $root \leftarrow$ newVertex(root, 0, t)
    $V(\Delta_{evol}) \leftarrow V(\Delta_{evol}) \cup \{root\}$
  for $S_i \in C_t$ **do**
    $u \leftarrow$ getVertex(Δ_{evol},sid_i)
    if $\exists S'_j \in C_{t+\delta t}$ *and* $M[S_i, S'_j] = 1$ **then**
      S_i **survives**
      $v \leftarrow$ newVertex(survives, sid_i, $t + \delta t$)
      $V(\Delta_{evol}) \leftarrow V(\Delta_{evol}) \cup \{v\}$
      $E(\Delta_{evol}) \leftarrow E(\Delta_{evol}) \cup \{u, v\}$
    else
      if $\exists \{S'_k \cup ... \cup S'_j\} \subseteq C_{t+\delta t}, \sum_{r=k}^{j} M[S_i, S'_r] \geq \tau$ **then**
        S_i **splits**
        for $S'_r \in \{S'_k \cup ... \cup S'_j\}$ **do**
          $v \leftarrow$ newVertex(split, sid_r, $t + \delta t$)
          $V(\Delta_{evol}) \leftarrow V(\Delta_{evol}) \cup \{v\}$
          $E(\Delta_{evol}) \leftarrow E(\Delta_{evol}) \cup \{u, v\}$
      else
        if $\exists S'_j \in C_{t+\delta t}, \{S_i \cup ... \cup S_k\} \subseteq C_t, \sum_{r=i}^{k} M[S_r, S'_j] \geq \tau$ **then**
          S_i,..., S_k **merge**
          $v \leftarrow$ newVertex(merge, sid_j, $t + \delta t$)
          $V(\Delta_{evol}) \leftarrow V(\Delta_{evol}) \cup \{v\}$
          $E(\Delta_{evol}) \leftarrow E(\Delta_{evol}) \cup \{u, v\}$
        else
          if $\nexists S'_j \in C_{t+\delta t}$, $M[S_i, S'_j] \geq \tau$ **then**
            S_i **disappears**
            $v \leftarrow$ newVertex(disappear, sid_{-1}, $t + \delta t$)
            $V(\Delta_{evol}) \leftarrow V(\Delta_{evol}) \cup \{v\}$
            $E(\Delta_{evol}) \leftarrow E(\Delta_{evol}) \cup \{u, v\}$
          else
            if $\exists S'_j \in C_{t+\delta t}$, $M[S_i, S'_j] \geq \tau$ **then**
              if $\left|O_{S_i}\right| > \left|O_{S'_j}\right|$ **then**
                S_i **shrinks**
                $v \leftarrow$ newVertex(shrink, sid_j, $t + \delta t$)
                $V(\Delta_{evol}) \leftarrow V(\Delta_{evol}) \cup \{v\}$
                $E(\Delta_{evol}) \leftarrow E(\Delta_{evol}) \cup \{u, v\}$
              if $\left|O_{S_i}\right| < \left|O_{S'_j}\right|$ **then**
                S_i **expands**
                $v \leftarrow$ newVertex(expand, sid_j, $t + \delta t$)
                $V(\Delta_{evol}) \leftarrow V(\Delta_{evol}) \cup \{v\}$
                $E(\Delta_{evol}) \leftarrow E(\Delta_{evol}) \cup \{u, v\}$
  for $S'_j \in C_{t+\delta t}$ **do**
    if $\forall S_i \in C_t, M[S_i, S'_j] = 0$ **then**
      S'_j **appears**
      $v \leftarrow$ newVertex(appear, sid_i, $t + \delta t$)
      $V(\Delta_{evol}) \leftarrow V(\Delta_{evol}) \cup \{v\}$
      $E(\Delta_{evol}) \leftarrow E(\Delta_{evol}) \cup \{root, v\}$

Algorithm 2: Apriori-based approach to detect frequent sequence evolution

Input: Δ_{evol}: a Graph Evolution Cluster, σ:minimum support
Output: $H_1, ..., H_k$ a set of frequent subgraphs of cardinality 1 to k

```
1  begin
2      H_1 ← detect all frequent paths with one vertex in Δ_evol
3      k ← 2
4      while H_{k-1} ≠ ∅ do
5          H_k ← ∅
6          CH_k ← candidate − gen(H_{k-1})
7          for g ∈ CH_k do
8              SI ← path-isomorphism(g, Δ_evol)
9              g.count ← |SI|
10             if g.count ≥ σ ∧ g ∉ H_k then
11                 H_k ← H_k ∪ g
12         k ← k + 1
```

This algorithm also generates Δ_{evol} incrementally.

In our approach we resort to a matrix M that stores in each cell the similarity value (computed according Definition 3.1.8) between two clusters in different timestamps. In this matrix, the lines represent the clusters that belong to C_t and the columns represent the clusters that belong to $C_{t+\delta t}$. For example, $M[S_i, S'_j]$ corresponds to the Jaccard similarity of the cluster $S_i \in C_t$ to cluster $S'_j \in C_{t+\delta t}$.

While the matrix M is calculated, the Algorithm 1 also generates Δ_{evol}. For each cluster $S_i \in C_t$, the algorithm gets the vertex u on Δ_{evol} that corresponds to the cluster S_i. If $\exists S'_j \in C_{t+\delta t}$, such that $M[S_i, S'_j] >= \tau$, then the evolution of S_i to S'_j can be survive, split, merge, disappear, shrink or expand. A vertex v is created to represent S'_j and an edge $\{u, v\}$ is also added on E(Δ_{evol}). On the case of appear evolution pattern, a cluster $S'_j \in C_{t+\delta t}$ appears if $\forall S_i \in C_t, M[S_i, S'_j] = 0$. Therefore, a vertex v is created for S'_j and an edge $\{root, v\}$ is added from *root* to v on E(Δ_{evol}).

Algorithm 1 receives as input M and detects each cluster evolution pattern from time t to time $t + \delta t$. The complexity of this naive algorithm is $O(n^2)$, assume that n is the order of the number of clusters. This processing can be costly, moreover, it needs to execute the clustering algorithm from scratch. The next chapter presents a new approach which detects cluster evolution using operations insert, delete or update of moving objects trajectories at each δt units of time, without recomputing all clusters from scratch. However, the approach presented in the next chapter is not able to detect frequent sequence(s) of evolution patterns.

Tabela 3 – Number of trajectories and clusters for Taxi dataset from Fortaleza, Brazil for the observed timestamps in these experiments according to Table 4

Timestamp	#Trajectories	#clusters
20:00	1314	18
21:00	1308	27
22:00	1306	38

A drawback of this naive approach is to define an adequate value for τ that represents the similarity among the clusters. Anyway, this algorithm is useful for our study, since it represents a brute force approach and reveal the challenges we faced in our work. The next section presents our experimental evaluation using this naive approach.

After generating Δ_{evol}, we need to find all frequent sequence evolution patterns with cardinality varying on 1 to k. This is the same of discovering the frequent paths on Δ_{evol}. For that, we used apriori-based approach discussed on (JIANG *et al.*, 2013). This algorithm is presented on Algorithm 2. Observe that we can identify the frequent pattern sequences involving three or more time instances using $k \geq 3$.

The basic Apriori-based algorithm is presented (AGRAWAL *et al.*, 1994). In line 6 all frequent $(k-1)$ path are used to generate k path candidates. If any of the $k-1$ candidate path are not frequent, then the algorithm safely prunes the candidates. For each candidate g, the algorithm keeps on *SI* data structure all the isomorphic path to g on Δ_{evol} (line 8). If the number of isomorphic path is greater than the support σ, g is a candidate of frequent path or sequence of evolution patterns. So that, g is stored on F_k.

There are many approaches to efficiently find the isomorphic path to g on Δ_{evol}. They are also present on (JIANG *et al.*, 2013).

3.3 Experimental Evaluation

The dataset used in the experiments was related to vehicle trajectories of a private Taxi fleet from the city of Fortaleza in Brazil. It contains for each vehicle a collection of timestamped positions (latitude and longitude) obtained from a GPS device. Each vehicle represents one moving object in our experiments. In these experiments we run the density based clustering algorithm DBSCAN (ESTER *et al.*, 1996) from scratch at every δt units of time. We used δt equals to one hour, the DBSCAN parameters are $eps = 50m$ and $minPoints = 5$. We defined τ equal to 0,5.

Tabela 4 – Some clusters evolution patterns for a Taxi Fleet dataset from Fortaleza, Brazil

Matrix Element(s)	**Evolution Pattern**
Evolution from 20:00 to 21:00	
$M[S_1, S'_1]$=1	S_1 survives
$M[S_2, S'_3]$=0,75, $\forall S_i \in C_{20:00}$ $i \neq 2$, $M[S_i, S'_3] = 0$ and $\lvert O_{S_2}\rvert > \left\lvert O_{S'_3}\right\rvert$	S_2 decreases
$M[S_5, S'_{17}]$=1	S_5 survives
$M[S_9, S'_{13}]$=0,46 and $M[S_{11}, S'_{13}]$=0,1	S_9 and S_{11} merge
$\forall S_i \in C_{21:00}$, $M[S_i, S'_{25}] = 0$	S'_{25} appears
Evolution from 21:00 to 22:00	
$M[S_1, S'_3]$=1	S_1 survives
$M[S_2, S'_4]$=0,83, $\forall S_i \in C_{21:00}$ $i \neq 2$, $M[S_i, S'_4] = 0$ and $\lvert O_{S_2}\rvert < \left\lvert O_{S'_4}\right\rvert$	S_2 increases
$\forall S'_j \in C_{22:00}$, $M[S_5, S'_j]$=0	S_5 disappears
$M[S_{12}, S'_7]$=0,06 and $M[S_{12}, S'_{15}]$=0,62	S_{12} splits
$M[S_{13}, S'_{16}]$=0,57 and $M[S_{13}, S'_{19}]$=0,14	S_{13} splits
$M[S_{15}, S'_{16}]$=0,05 and $M[S_{15}, S'_{19}]$=0,57	S_{15} splits
$M[S_{24}, S'_{25}]$=1	S_{24} survives
$\forall S_i \in C_{21:00}$, $M[S_i, S'_1] = 0$	S'_1 appears

The number of moving objects trajectories used on the first three executions of DBSCAN is described on Table 3 as well as the number of clusters found. Note that the number of trajectories is almost the same in all cases, however, there are new, deleted and up-to-date moving object trajectories between two different timestamps. By using our approach, we can also discover how long a mobility pattern takes during the observed time interval.

In these experiments, it is our goal to detect for all clusters on every δt units of time their evolution pattern. Moreover, to discover all the frequent sequences of cluster evolution pattern. We believe that a sequence of evolution can describe a mobility pattern as we did before for modeling traffic jam in the beginning of this chapter.

In the Table 4 we present some cases of cluster evolution patterns found and the result covers all the patterns discussed before. Note that the clusters do not remain the same for different timestamps. For example, the cluster $S_5 \in C_{21:00}$ disappears and the cluster S'_1 appears on $C_{22:00}$. We did not show all the elements of the two obtained evolution matrices, because it would not be trivial to visualize.

The results on Table 5 present the frequent sequence of cluster evolution pattern found by our approach using k=5 and σ=20. The results suggest that evolution patterns and their frequency can be effectively obtained through the proposed Graph Evolution Cluster over Time Δ_{evol} from moving objects dataset. Observe that merge and split evolutions are more frequently than others. Both cluster evolutions capture the evolvements and interaction of clusters over time

Tabela 5 – Some Frequent Sequence Cluster Evolution Patterns for Taxi Fleet dataset from Fortaleza, Brazil

Frequent Sequence Evolution Pattern	**Frequency**
k=1	
SHRINKS	331
EXPANDS	317
SPLIT	206
MERGE	178
k=2	
EXPANDS–SHRINKS	117
MERGE–MERGE	160
SPLIT–SPLIT	150
EXPANDS–EXPANDS	90
k=3	
MERGE–MERGE–MERGE	100
SPLIT–SPLIT–MERGE	62
SHRINKS–EXPANDS–SHRINKS	51
EXPANDS–SHRINKS–EXPANDS	45
EXPANDS–EXPANDS–SHRINKS	35
k=4	
MERGE–MERGE–MERGE–MERGE	46
SPLIT–SPLIT–SPLIT–MERGE	45
MERGE–MERGE–MERGE–EXPANDS	36
MERGE–MERGE–MERGE–SPLIT	34
SPLIT–SPLIT–MERGE–MERGE	33
k=5	
SPLIT–SPLIT–MERGE–MERGE–MERGE	41
SPLIT–SPLIT–SPLIT–MERGE–SPLIT	27
MERGE–MERGE–MERGE–MERGE–EXPANDS	26
SPLIT–SPLIT–SPLIT–SHRINKS–SPLIT	24
SPLIT–SPLIT–MERGE–MERGE–SPLIT	23

as we aimed. Moreover, our approach does not reduce to counting for all the possible sequence of cluster evolution patterns, how many times a pattern occurs. We prune some possible sequence, this is less computationally expensive.

Figure 5 presents another experiment using the same dataset of taxi in Fortaleza. In this experiment, we show for three consecutive time windows some of the cluster evolution patterns captured by our proposed Algorithm 1. Each cluster is represented by a single color. Not all evolution patterns found are presented here to ease viewing. But we can easily see, the differences between the clusters by comparing the three time windows. For each time window, we executed the DBSCAN algorithm from scratch.

In the next section, we propose CUTiS which does not execute the clustering algo-

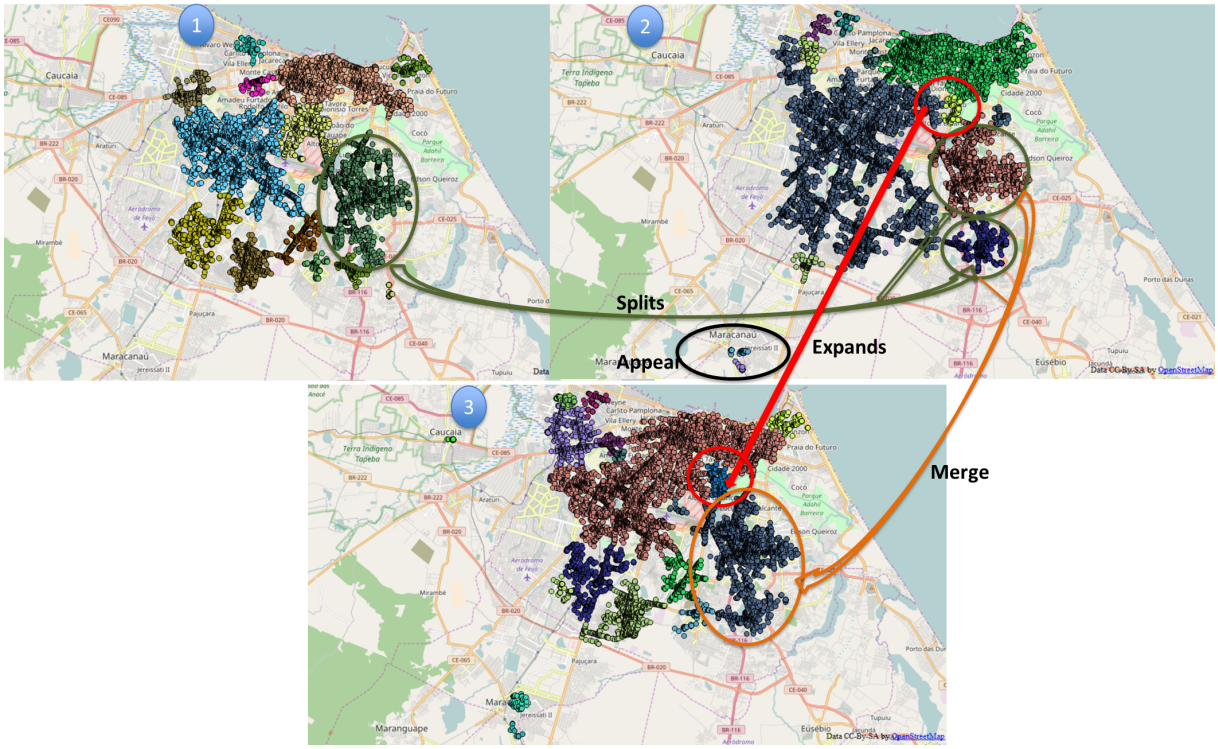


Figura 5 – Example of detecting cluster evolution pattern for three consecutive time windows

rithm from scratch and captures the clusters evolution patterns online. However, CUTiS works for trajectory data and it can not capture the frequencies of evolution patterns as the approach presented in this chapter. So even, this chapter proposes a naive approach to detect cluster evolution, it is addressed in the moving objects position and it captures the frequent sequence of evolution patterns which is new in the literature.

3.4 Discussion

In this chapter, we worked on a model to describe cluster evolution patterns, how to detect them and discover the frequent sequence of cluster evolution. We claim that some mobility patterns can be described as a sequence of cluster evolutions. As we described the traffic jam pattern as a sequence $(appear, merge, expand|shrink, disappear)$. In this way, we need to keep tracking of the evolution of the moving objects to achieve our goal.

Our proposal is a naive approach which executes DBSCAN clustering algorithm from scratch every δt units of time, which can be time-consuming for very large databases. These results were published in (SILVA *et al.*, 2014b). We extended this work for trajectory data by proposing CUTiS. Our framework CUTiS automatically detects cluster evolution patterns while tracking moving object trajectories. CUTiS is given in details in the next Chapter.

4 CUTIS: CLUSTERING ON TRAJECTORY DATA STREAM

In this Chapter we present our solution called CUTiS for the **Challenges 1** and **2** presented in the Section 1.2. The problem, in general lines, corresponds to online discovery spatio-temporal patterns in Euclidean Space, while efficiently maintain and detect the evolution of these patterns as time evolves.

4.1 Problem Statement

In this section we provide the problem definition and the problem statement. We start by providing the basic notions related to trajectories.

Definition 4.1.1 (**Trajectory**). The movement of a moving object o_j can be described in terms of a continuous function $M_j : R_{\geq 0} \rightarrow R^2$ from the domain of real positive numbers, representing time instants, to a 2D space. Given an object o_j and a time interval $[tBegin, tEnd]$, a trajectory TR is thus the restriction of the movement M_j to the given time interval.

Commonly, in mobility applications the continuous movement of an object is not completely known. As such, trajectories are often given by means of a finite set of time-stamped positions, called trajectory points or *samples*.

Definition 4.1.2 (**Trajectory point**). Given a moving object o_j, a trajectory point, or trajectory sample, of a trajectory TR_j is a tuple (x, y, t), where $TR_j(t) = (x, y)$ is the position of object j at time t (called the timestamp of the trajectory point).

Definition 4.1.3 (**Trajectory track**). Given a temporally ordered sequence $< t_1, \cdots, t_n >$ of timestamps, the track of a trajectory TR_j for the given timestamps is the temporally ordered sequence of trajectory points $< p_1, \cdots, p_n >$, where $p_i = (x_i, y_i, t_i)$ and $(x_i, y_i) = TR_j(t_i)$.

For the sake of brevity, hereinafter we will use the term *trajectory* in place of *trajectory track*. From the definition of trajectory we can derive the definition of *sub-trajectory*.

Definition 4.1.4 (**Sub-trajectory**). Let us consider the trajectory TR_j. Let us also suppose that from TR_j we extract the *substring* $ST =< p_a, ..., p_b >$, where $1 \leq a \leq b \leq n$. We define ST to be a *sub-trajectory* of TR_j.

At this point we can introduce the concept of *input stream*.

Definition 4.1.5 (**Input stream**). Let $i = [t, t + \delta t]$ be a *time window* (temporal interval): we define the set $I_i = \{(o_1, ST_{1,i}), (o_2, ST_{2,i}),...,(o_n, ST_{n,i})\}$ to be an *input stream* that spans the time window i, where $ST_{j,i}$ represents the sub-trajectory associated with a moving object o_j that spans i.

Hereinafter we assume that there is no overlap between temporal intervals associated with different input streams.

We aim to track moving objects, as well as to discover sub-trajectory clusters incrementally, while, at the same time, avoid to execute our clustering algorithms from scratch at each time window. Since we are considering a stream of (dynamic) data, each sub-trajectory is manipulated in our proposed solutions according to one of the following operations : (i) a moving object can appear in the system (thus, the object is inserted into the system), (ii) a moving object can disappear from the system (thus, the object is deleted from the system), and (iii) moving objects may update their sub-trajectories during the time window. As such, these operations may affect the composition of existing clusters.

Definition 4.1.6 (**Problem Statement**). For each time window i, when an input stream $I_i = \{(o_1, ST_{1,i}), (o_2, ST_{2,i}), ..., (o_n, ST_{n,i})\}$ is received by the system, the goal is to track moving objects and update clusters by avoiding to re-compute the clusters from scratch at each time window. Finally, sub-trajectory clusters should be discovered by taking advantage of movement similarities between objects, thus taking into account time and space.

In order to discover sub-trajectory clusters having any kind of shape, we recur to *density-based* clustering (ESTER *et al.*, 1996) techniques.

4.2 CUTiS: Clustering on Trajectory Data Stream

In this section we present CUTiS, an algorithm that addresses the problem of *online discovery of sub-trajectory clusters*, as well as the problem of keeping track of the *evolution* of clusters over time, by tracking moving objects across consecutive time windows.

CUTiS, which was first proposed in (SILVA *et al.*, 2016b; SILVA *et al.*, 2016a), is structurally arranged in four phases (as schematized also in Figure 6):

1. **Application of a distance function**: To measure the distance between the sub-trajectories of two moving objects at time window i, that is, $ST_{k,i}$ and $ST_{j,i}$, we implement the *synchro-*

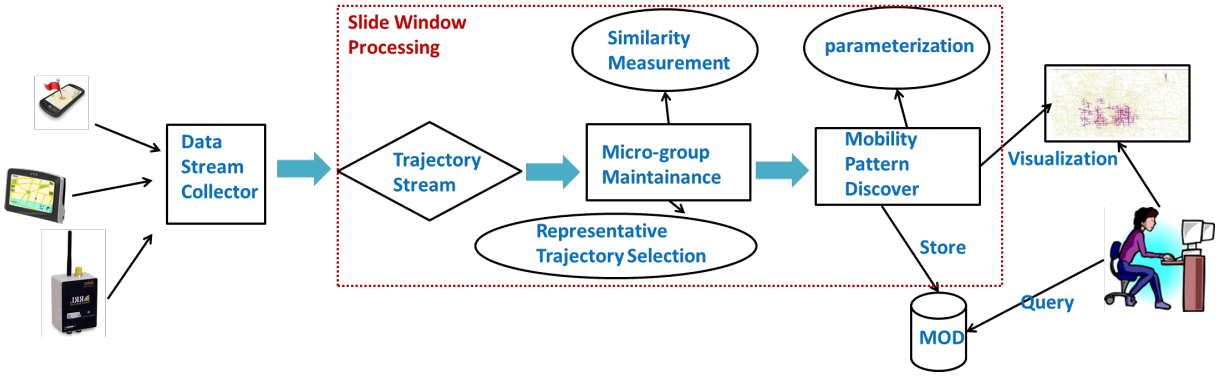


Figura 6 – CUTiS' framework structural overview.

nous Euclidean distance (Section 4.2.1), which accounts for time and space. We highlight that CUTiS can be used with any distance function suitable for pairs of trajectories.

2. **Selection of representative trajectories**: given a time window i, for each group of moving objects and their related sub-trajectories, i.e., the input stream $I_i = \{(o_1, ST_{1,i}), (o_2, ST_{2,i}), ..., (o_m, ST_{m,i})\}$, we define the *representative trajectory* to be a pair made up by a moving object and a sub-trajectory which "better represents" the behavior of I_i. To find a suitable representative sub-trajectory $(o_j, ST_{j,i})$, CUTiS uses two metrics: (i) the number of moving objects that have their sub-trajectories similar to $ST_{j,i}$ and (ii) a Gaussian kernel function that estimates the representativeness of $ST_{j,i}$. We delegate the discussion of this matter to Section 4.2.2.
3. **Micro-groups maintenance**: We define a new data structure, called *micro-group*, that serves the purpose of characterizing small and dense groups of moving objects moving similarly with respect to some representative trajectory (during some time interval). Intuitively, the concept of micro-group corresponds to a group of entities that move consistently together in space for some time – for example, a group of people riding and sharing the same vehicle.

   In this work we propose an algorithm that is able to detect micro-groups, as well as incrementally maintain and capture their evolution patterns over time. We finally propose a density-based cluster discovery method that is able to merge different micro-groups. We refer the reader to Section 4.2.3 for more details.
4. **Discover Sub-trajectory Clusters**:

   Micro-groups rely on the concept of *density*; as such, clusters of sub-trajectories can be discovered from micro-groups by means of this property – for example, when certain conditions apply, merging different micro-groups yields a density-based cluster of sub-

Notation	Description
ε	Distance threshold.
τ	Size or density threshold.
ρ	Representativeness threshold.
$i = [t, t+\delta t]$	A time window.
I_i	Input stream associated with time window i.
o_i, o_j, o_k	Moving objects.
$ST_{j,i}$	Sub-trajectory of o_j limited to the time window i.
O_i	Set of moving objects that are *active* during the time window i.
g_i, g_j, g_k	Micro-groups.
$R_{g_i}^{traj}$	Representative trajectory associated with g_i.
$object[R_{g_i}^{traj}]$	Moving object that expresses the representative trajectory of g_i.
$traj[R_{g_i}^{traj}]$	Sub-trajectory that represents the representative trajectory of g_i.
G_i	Set of micro-groups at time window i.
c	Cluster.
C	Set of clusters.
$R_c^{\{traj\}}$	Representative trajectories associated with the cluster c.

Tabela 6 – Notation used throughout the chapter

trajectories. Moreover, the usage of micro-groups allows CUTiS to be more efficient than other approaches – indeed, in the experimental evaluation (Section 4.6) we show how CUTiS is more efficient than having to run some clustering algorithm from scratch at every time window (to this end, we use DBSCAN (ESTER *et al.*, 1996) as the baseline). For more details about this phase, we refer the reader to Section 4.3.

In order to facilitate the readability, Table 6 reports the notation used throughout the rest of the chapter.

4.2.1 Distance Function

In this Section we present the distance function used by CUTiS to measure the distance between pairs of trajectories. The function has two levels, where one level deals with space and time, while the other one deals with the angle of movement.

Definition 4.2.1 (**Spatio-Temporal Distance**). Given a time window $i = [t, t+\delta t]$, the *Spatio-Temporal Distance* between the sub-trajectories of two moving objects, $ST_{j,i}$ and $ST_{k,i}$, is computed by means of the synchronous Euclidean distance (NANNI; PEDRESCHI, 2006) $D_{ST_{j,i},ST_{k,i}}$:

$$dist_{\lambda}(ST_{j,i}, ST_{k,i}) = \frac{\int_{t}^{t+\delta t} D_{ST_{j,i},ST_{k,i}}(t)\,\mathrm{d}t}{\delta t} \tag{4.1}$$

Let $D_{ST_{j,i},ST_{k,i}}(t)$ be the distance function applied on the sub-trajectories $ST_{j,i}$ and $ST_{k,i}$ at time t. Let us also consider the time interval $[t, t+\delta t]$ and let n be the number position updates received during the time interval. Then, we can rewrite Equation 4.1 as:

$$dist_{\lambda}(ST_{j,i}, ST_{k,i}) = \frac{\sum_{r=1}^{n-1} \int_{t_r}^{t_{r+1}} D_{ST_{j,i},ST_{k,i}}(t)\,\mathrm{d}t}{\delta t}, \tag{4.2}$$

where each t_r is the timestamp when at least one of the moving objects report its position. We note that the above equation requires that sub-trajectories possess at least 2 useful samples, since CUTiS relies on approximated linear interpolation to estimate the position of a trajectory at a given time instant.

The Euclidean distance between two objects, where objects move between consecutive timestamps according to some linear function of time, can be defined according to the definition provided by (FRENTZOS *et al.*, 2007a), i.e.:

$$D_{ST_{j,i},ST_{k,i}}(t) = \sqrt{at^2 + bt + c}. \tag{4.3}$$

Here, a, b, and c represent the factors of the trinomial ($a \geq 0$). As proved in (FRENTZOS *et al.*, 2007b), the integral can be computed in $O(1)$ and can be efficiently computed by means of the Trapezoid Rule (FRENTZOS *et al.*, 2007b):

$$d_{\lambda}(ST_{j,i}, ST_{k,i}) \approx \frac{\frac{1}{2}\sum_{r=1}^{n-1} [D_{ST_{j,i},ST_{k,i}}(t_r)] * (t_{r+1} - t_r)}{\delta t} + \frac{\frac{1}{2}\sum_{r=1}^{n-1} [D_{ST_{j,i},ST_{k,i}}(t_{r+1})] * (t_{r+1} - t_r)}{\delta t} \tag{4.4}$$

We also note that the Trapezoid Rule guarantees theoretical bounds for what is related to the approximation error. At this point we can introduce the concept of *angle distance*.

Definition 4.2.2 (Angle Distance). The *angle distance* between two sub-trajectories, $ST_{j,i}$ and $ST_{k,i}$, during a time window $i = [t, t+\delta t]$ is based on the *angle* formed by the line segments making up the sub-trajectories. Since time is discretized, and multiple samples may be contained

within the considered time interval, the distance is computed by summing the outputs given by a function D_θ over the line segments making up the sub-trajectories:

$$d_\theta(ST_{j,i}, ST_{k,i}) \approx \frac{\sum_{r=1}^{n-1} D_\theta[(ST_{j,i}(t_r), ST_{j,i}(t_{r+1})), (ST_{k,i}(t_r), ST_{k,i}(t_{r+1}))]}{\delta t} \quad (4.5)$$

where $(ST_{j,i}(t_r), ST_{j,i}(t_{r+1}))$ and $(ST_{k,i}(t_r), ST_{k,i}(t_{r+1}))$ represent the line segments of $ST_{j,i}$ and $ST_{k,i}$, respectively, while t_r represents the timestamp when an update holds.

The definition of D_θ is based on the one provided by (LEE *et al.*, 2007), and it expresses the orientation difference between two line segments. More precisely, let $L_j = (ST_{j,i}(t_r), ST_{j,i}(t_{r+1}))$, $L_k = (ST_{k,i}(t_r), ST_{k,i}(t_{r+1}))$ and θ be the smaller intersecting angle between L_j and L_k. The intuition behind is to choose a line segment, let's say L_j, and discover the travelled length during the rotation. Then:

$$D_\theta(L_j, L_k) = \begin{cases} \left|L_j\right| * \sin(\theta), & \text{if } 0^\circ \leq \theta < 90^\circ. \\ \left|L_j\right|, & 90^\circ \leq \theta \leq 180^\circ. \end{cases} \quad (4.6)$$

The angle θ can be calculated by means of simple vector operations. Indeed, as L_j and L_k represents line segments, we have that θ can be determined as:

$$\cos(\theta) = \frac{\vec{L_j}\vec{L_k}}{\|\vec{L_j}\|\|\vec{L_k}\|} \quad (4.7)$$

At this point we can finally define the distance measure between two sub-trajectories, $ST_{j,i}$ and $ST_{k,i}$:

$$distance(ST_{j,i}, ST_{k,i}) = \omega_\theta * d_\theta(ST_{j,i}, ST_{k,i}) + \omega_\lambda * d_\lambda(ST_{j,i}, ST_{k,i}) \quad (4.8)$$

We note that the weights ω_θ and ω_λ must be chosen according to the scenario considered – indeed, d_θ and d_λ may be characterized by different magnitudes; for the purposes of this work, we assume that ω_θ and ω_λ are both equal to 0.5.

4.2.2 Choosing a representative trajectory

Given a group of sub-trajectories, a representative trajectory should describe their overall movements. The definition that follows try to capture this aspect.

Definition 4.2.3 (**Representative trajectory**). Let us consider the input stream $I_i = \{(o_1, ST_{1,i}), (o_2, ST_{2,i}), .., (o_m, ST_{m,i})\}$. Let also ρ be a *representativeness* threshold, σ be a *standard deviation*, ε a *distance* threshold and τ a *minimum size/density* threshold. Then, we define $(o_j, ST_{j,i})$ to be *representative* trajectory of I_i *if and only if* the following conditions hold:

1. $\forall (o_k, ST_{k,i}) \in I_i$, $voting(ST_{j,i}, ST_{k,i}) = e^{-\frac{distance^2(ST_{j,i}, ST_{k,i})}{2\sigma^2}} > \rho$
2. $N_\varepsilon(o_j) = \{(o_k, ST_{k,i}) \in I_i \mid distance(ST_{j,i}, ST_{k,i}) \leq \varepsilon\}$, then $\left|N_\varepsilon(o_j)\right| \geq \tau$

Our *voting* function (condition 1 in Definition 4.2.3) is based on the voting function defined in (PANAGIOTAKIS *et al.*, 2012), which uses the Gaussian function to quantify the representativeness of trajectory line segments. In this work, *voting* quantifies the representativeness of a sub-trajectory with respect to a group of sub-trajectories. In order to be representative, the "voting" of a sub-trajectory should be significant (i.e., greater than the ρ threshold). As such, choosing a proper value for ρ is important; in our work, we deemed reasonable to pick up ρ according to the maximum distance *allowed* between any sub-trajectory and its representative. Finally, in CUTiS we choose representative trajectories according to the notion of *density property* (dense neighborhood, condition 2 in Definition 4.2.3).

We finally spend some words about the relationship between *distance* and *voting*: on the one hand, if *distance* is close to zero, then *voting* is close to its maximum value. In practical terms, this means that if $ST_{j,i}$ is very close (in time and space) to $ST_{k,i}$, then $(o_j, ST_{j,i})$ is a candidate to be the representative of $ST_{k,i}$. On the other hand, when "distance" gets high, then "voting" gets close to its minimum value, which in turn implies that $ST_{j,i}$ is far away from $ST_{k,i}$: as such, $ST_{k,i}$ cannot be represented by $ST_{j,i}$.

4.2.3 Micro-groups definition, discovery and maintenance

In the following we present a new structure that builds on the concept of representative trajectory to represent a group of *co-moving* sub-trajectories: we call this structure *micro-group*. Subsequently, we provide a set of algorithms whose purpose is to discover and maintain micro-groups over time.

We start by providing the definition of micro-group.

Definition 4.2.4 (**Micro-group**). Let $I_i = \{(o_1, ST_{1,i}), (o_2, ST_{2,i}), ..., (o_m, ST_{m,i})\}$ be an input stream at time window i; let also O_i be the set of moving objects in I_i, ε be a distance th-

reshold, τ be a minimum density threshold, and ρ be a representativeness threshold. Then, a *micro-group* g is defined as a subset of O_i that satisfies the following property:

$\exists o_j \in g$, such that $R_g^{traj} = (o_j, ST_{j,i})$ is the representative trajectory of g with respect to ε, τ and ρ.

We note that a micro-group may be seen as the most primitive form of a cluster of sub-trajectories. However, in CUTiS we differentiate between the former and the latter by means of the notion of *density-based* clustering (ESTER *et al.*, 1996). Accordingly, in the following we introduce a set of definitions that will be subsequently used to define what is a cluster of sub-trajectories.

Definition 4.2.5 (**Directly Density Reachable**). Let G_i be the set of micro-groups at time window i. A micro-group $g_j \in G_i$ is *directly density reachable* from a micro-group $g_k \in G_i$ w.r.t. ε and τ, if $\exists o_j \in g_j, \exists o_k \in g_k$, such that the following conditions hold:

1. $distance(ST_{j,i}, ST_{k,i}) \leq \varepsilon$.
2. $\left|N_\varepsilon(o_j)\right| \geq \tau$, where $N_\varepsilon(o_j) = \{o'_j \in G_i \mid distance(ST_{j,i}, ST_{j',i}) \leq \varepsilon\}$.
3. $|N_\varepsilon(o_k)| \geq \tau$, where $N_\varepsilon(o_k) = \{o'_k \in G_i \mid distance(ST_{k,i}, ST_{k',i}) \leq \varepsilon\}$.

Definition 4.2.6 (**Density Reachable**). A micro-group g_i is *density reachable* from g_k w.r.t. ε and τ, if there is a chain of micro-groups $\{g_{s_1}, ..., g_{s_n}\}$, where $g_{s_1} = g_i$, $g_{s_n} = g_k$ such that $g_{s_{j+1}}$ is directly density reachable from g_{s_j}.

Definition 4.2.7 (**Density Connected**). A micro-group g_i is density connected to g_k w.r.t. ε and τ, if there is a micro-group g_j such that g_i and g_k are density reachable from g_j w.r.t. ε and τ.

At this point, we are able to define a sub-trajectory cluster in CUTiS. Intuitively, a cluster is defined to be a set of density-connected micro-groups. *Outliers* represents simply the set of sub-trajectories in the input stream at time window i, I_i not belonging to any of its clusters.

Definition 4.2.8 (**Cluster of sub-trajectories**). Let $G_i = \{g_1, ..., g_n\}$ be a set of micro-groups at time window i. Then, a *cluster* C w.r.t. ε and τ is a non empty subset of G_i that satisfies the following conditions:

1. $\forall g_j, g_k$ if $g_j \in C$ and g_k is *density reachable* from g_j w.r.t. ε and τ, then $g_k \in C$.
2. $\forall g_j, g_k \in C$, g_j is *density connected* to g_k w.r.t. ε and τ.

We maintain micro-groups over time by tracking their evolution. We remember that typical evolution patterns are *appear*, *disappear*, *split*, *survive*, and *merge* (SPILIOPOULOU *et*

al., 2006). We cover all these kinds of patterns during the micro-group maintenance phase with the exception of merge, since we see the merging of two or more micro-groups as a sub-trajectory *cluster*. Hence, a sub-trajectory *cluster* may have multiple representative trajectories. This is the major difference between a *cluster* and a *micro-group*.

Once we have the definition of micro-group in place, we need to provide a method that is able to create micro-groups. The idea is the following: we start by picking up, randomly, a *candidate* representative trajectory among the trajectories associated with the core objects (similarly to DBSCAN). Subsequently, the candidate is evaluated by means of a *voting* procedure that is carried on by the core objects: if the candidate is deemed to be enough representative, the algorithm terminates and returns the candidate, otherwise it goes on to consider the next candidate among the remaining ones. Algorithm 3 illustrates our strategy.

Algorithm 3: *initializeMicroGroups*

Input:
- Input stream at time interval $i = [t, t+\delta t]$, I_i.
- Distance threshold ε.
- Size threshold τ.
- Representativeness threshold ρ.

Output: Set of micro-groups, G_i.

```
1  begin
2      G_i ← ∅; g ← ∅;
3      Mark all the sub-trajectories in I_i as unvisited;
4      while ∃(o_j, ST_{j,i}) ∈ I_i s.t. (o_j, ST_{j,i}) is unvisited do
5          mark (o_j, ST_{j,i}) as visited;
6          N_ε^{o_j} ← getNeighbourhood(I_i, ε, o_j);
7          if |N_ε^{o_j}| -1 ≥ τ then
8              foreach (o_k, ST_{k,i}) ∈ I_ε not visited do
9                  vote ← voting(ST_{j,i}, ST_{k,i});
10                 if vote > ρ then
11                     g ← g ∪ {o_k};
12                     mark (o_k, ST_{k,i}) as visited;
13             R_g^{traj} ← (o_j, ST_{j,i});
14             G_i ← G_i ∪ {g}; g ← ∅;
15         else
16             mark ST_{j,i} as ungrouped;
17     foreach ST_{j,i} ∈ I_i marked as ungrouped do
18         mark o_j as an outlier;
19     return G_i;
```

The algorithm starts by randomly picking up an unvisited sub-trajectory $(o_j, ST_{j,i}) \in I_i$, and checks if it can be used as a representative trajectory: if $(o_j, ST_{j,i})$ has enough neighborhood density (according to the ε radius and the τ threshold - lines 6 - 7), then a new

micro-group "g" is created; in such a case, the algorithm goes on to expand g by inserting unvisited objects in I_i that votes to be represented by $(o_j, ST_{j,i})$ (lines 8 - 12). If, conversely, $(o_j, ST_{j,i})$ has not enough neighborhood density, then o_j is marked as *ungrouped* (lines 15 - 16), thus indicating that it can be represented by another representative trajectory, or it is an outlier (lines 17 - 18).

Once all the suitable elements were considered for grouping, $(o_j, ST_{j,i})$ is classified as g's representative trajectory (line 13) and the algorithm goes on to check if it can create other micro-groups (line 4). Once no more unvisited elements are present in I_i, the algorithm terminates by returning the set of micro-groups found during its execution (line 19).

Overall, the initialization cost of Algorithm 3 is $O(n^2)$, "led" by the cost in Lines 4 - 16, where n is the number of moving objects. However, we note that this step needs to be carried out only once, since each micro-group can be dynamically maintained across different time windows (we outline the presentation of the maintenance procedure in Algorithm 4). We also note that the initialization cost associated with Algorithm 3 cannot be further reduced, since we assume that moving objects are continuously updating their positions over time.

Since CUTiS deals with input streams, another key contribution of this work consists in providing an algorithm that *maintains* micro-groups over time.

The main challenges associated with a maintenance algorithm are the following: (i) we need to check for each micro-group whether its representative trajectory is still valid in the next time window with respect to the previous ones – if this is not the case, then the micro-group shall disappear or split; (ii) we need to track moving objects whose sub-trajectories are likely to join some micro-group – for instance, outliers, new objects entering the system, and objects that migrate from their micro-group to other ones; (iii) for what is related to the remaining cases, we need some kind of process that allows the creation of new micro-groups. Algorithm 3 is not deterministic, hence the interest of devising a maintenance algorithm is two-fold: first, running from scratch the initialization algorithm may lead to totally different micro-groups with respect to previous time windows. Secondly, without a maintenance algorithm, we would not be able to track the evolution of micro-groups over time.

As such, CUTiS recurs to a maintenance phase to capture the evolution of patterns over time, which in turn provides insights about the nature of changes affecting clusters (groups) of objects. Algorithm 4 presents the pseudocode detailing the maintenance phase.

The algorithm takes in input a micro-group at a time – for the purposes of the

Algorithm 4: Incremental Maintenance of a micro-group

Input:
- Input stream at time interval $i = [t, t+\delta t]$, I_i.
- A micro-group g_{i-1}.
- Distance threshold ε.
- Size threshold τ.
- Representativeness threshold ρ.
- The set of moving objects that must be removed from g_{i-1}, $O_{g_i}^{old}$.
- The set of moving objects that must update their current status in the transition from g_{i-1} to g_i, $O_{g_i}^{upd}$.

Output: A set of micro-groups.

1 **begin**
2 $g_i \leftarrow g_{i-1} - O_{g_i}^{old}$;
3 **foreach** $o \in O_{g_i}^{upd}$ **do** $update(o, g_i)$;
4 $(o_j, ST_{j,i}) \leftarrow R_{g_i}^{traj}$;
5 $N_\varepsilon^{o_j} \leftarrow getNeighborhood(I_i, \varepsilon, o_j)$;
6 **if** $(|N_\varepsilon^{o_j}| \geq \tau) \wedge (\forall o_k \in N_\varepsilon^{o_j}.voting(ST_{j,i}, ST_{k,i}) > \rho)$ **then** // g_i survives
7 **foreach** $o_k \in \{g_i \setminus N_\varepsilon^{o_j}\}$ **do**
8 **if** $voting(ST_{j,i}, ST_{k,i}) \leq \rho$ **then**
9 $g_i \leftarrow g_i - \{o_k\}$;
10 **if** $\exists g_z \in G_i$ *s.t.* $voting(traj[R_{g_z}^{traj}], ST_{k,i}) > \rho$ **then** $g_z \leftarrow g_z \cup \{o_k\}$;
11 **else** $setOutlier(o_k, ST_{k,i})$;
12 **else**
13 **if** $|g_i| \geq \tau$ **then** // g_i splits
14 $I_i^{g_i} \leftarrow getInputStream(g_i)$;
15 **return** $initializeMicroGroup(I_i^{g_i}, \varepsilon, \tau, \rho)$;
16 **else** // g_i disappears
17 **foreach** $o_k \in g_i$ **do** $setOutlier(o_k, ST_{k,i})$;
18 $g_i \leftarrow \emptyset$;
19 **return** g_i;

presentation, we assume that we are considering some micro-group g_{i-1} – and two sets of objects: $O_{g_i}^{old}$, which refers to the set of objects that must be removed from g_{i-1}, while the latter, and $O_{g_i}^{upd}$, which refers to the set of objects that have to update their current status. When a moving object stops sending its positions (because of a GPS connection failure, for instance), the moving object sub-trajectory positions should be predicted until a timeout is reached. In this way, we assume that $g_{i-1} \subseteq O_{g_i}^{old} \cup O_{g_i}^{upd}$ and $O_{g_i}^{old} \cup O_{g_i}^{upd} \subseteq g_{i-1}$, i.e., any moving object that is not removed from g_{i-1} has to update its current status.

Between lines 2 and 3 the algorithm computes the initial g_i, i.e., it removes from g_{i-1} the objects belonging to $O_{g_i}^{old}$, and updates the status of the objects belonging to $O_{g_i}^{upd}$. Next, the algorithm has to check whether $(o_j, ST_{j,i}) = R_{g_i}^{traj}$ is still the representative trajectory of g_i. To do this, the algorithm first determines the set of o_j's neighbors (according to the ε radius, line 5); then, it checks whether (i) the size of $N_\varepsilon^{o_j}$ is greater or equal than τ and (ii) the votes of the neighbors still support the representativeness of $(o_j, ST_{j,i})$ (line 6): if these two conditions

hold then g_i survives and $(o_j, ST_{j,i})$ is still deemed as the representative of g_i (w.r.t. τ, ε and ρ); also, the algorithm checks whether each moving object $o_k \in g_i$ can be still represented by $(o_j, ST_{j,i})$ (line 7): if this is not the case, then the algorithm checks whether o_k can be migrated to another micro-group or not (line 10); if this is possible, then o_k is deleted from g_i and migrated to another micro-group; otherwise o_k is considered as an outlier (lines 11). In the case that the conditions at line 6 do not hold, this means that the micro-group represented by g_i changed its overall behavior. as such, the possible outcomes are two: if g_i possess enough density (τ), then g_i is *split* in multiple micro-groups by means of *initializeMicroGroups* (lines 13 - 15). Conversely, g_i is removed from the system and its objects are considered outliers (Lines 16 - 18).

At this point we can detail the complexity behind Algorithm 4. Let m be the number of micro-groups at time window i, and n_i^u be the number of moving objects which require to update their current status at time window i. At micro-group level the cost of Algorithm 4 is $O(m * n_i^u)$, which is due to the operations carried on between lines 7 and 11. Since Algorithm 4 is executed for each micro-group, its total cost is equal to $O(m^2 * n_i^u)$. Overall, the number of micro-groups is usually much smaller than the number of moving objects ($m \ll n$, and even $n_i^u \ll n$); as such, executing Algorithm 4 is more convenient than rebuilding all the micro-groups from scratch.

We close this section by illustrating how new objects that enter the system are processed by CUTiS – we note that this represents a notable difference with respect to prior works (TANG *et al.*, 2012; CAO *et al.*, 2006; CHEN; TU, 2007; LI *et al.*, 2010b), since CUTiS offers a complete set of operations (i.e., insertions, deletions and updates) to handle data streams of moving objects. Algorithm 5 presents our strategy.

For each new moving object o_k (line 2), Algorithm 5 tries to find a micro-group g_i (Line 3) whose representative trajectory can represent the sub-trajectory of o_k – this condition holds whenever $voting(traj[R_{g_i}^{traj}], ST_{k,i}) > \rho$ (Line 4). Objects that do not belong to any micro-group may be able, togheter with the outliers, to create new micro-groups. As such, we execute again Algorithm (Line 10) providing in input I^{new}, deprived of the objects assigned to some micro-group, and the set of outliers. We note that the insertion of new moving objects may be able to preserve the existence of micro-groups that have to disappear or split as a consequence of deletions or updates; however, we leave this issue for future research.

The overall cost of Algorithm 5 is $O(n_{new} * m) + O((n_{new} + n_{out})^2)$, where $n_{new} = |I_{new}|$, m is the number of micro-groups and $n_{out} = |Outliers|$. In this way, the cost of Algorithm

Algorithm 5: Insertion of new moving objects

Input:
- The set of micro-groups at time window i, G_i.
- The representativeness threshold, ρ.
- The subset of the input stream I_i limited to the set of moving objects that entered the system, I_i^{new}.
- The subset of the input stream I_i limited to the set of moving objects that were labeled as outliers, $Outliers$.

Output: The set of micro-groups at time window i, G_i, updated.

```
1  begin
2      foreach {o_k, ST_{k,i}} ∈ I_i^new do
3          foreach g_i ∈ G_i do
4              vote ← voting(traj[R_{g_i}^traj], ST_{k,i});
5              if vote > ρ then
6                  g_i ← g_i ∪ {o_k};
7                  I_i^new ← I_i^new − (o_k, ST_{k,i});
8                  break;
9      if I_i^new ≠ ∅ then
10         G_i ← G_i ∪ {initializeMicroGroups({I_i^new ∪ Outliers}, ε, τ, ρ)};
11     return G_i;
```

3 is limited to the number of new moving objects and outliers considered. All in all, $m \ll n$, $n_{new} \ll n$, and even $(n_{new} + n_{out}) \ll n$, where n is the total number of moving objects.

4.3 Density-based Sub-trajectory Clustering

Existing clustering algorithms need to check the density connectivity of each object to cluster trajectories (ESTER *et al.*, 1996; ANKERST *et al.*, 1999); this, in turn, makes such computational approaches expensive. On the contrary, CUTiS exploits the density property underlying the concept of micro-group to simplify the clustering process and reduce the costs. We remind that a cluster of sub-trajectories in CUTiS can be a micro-group itself (Section 4.2.3). Moreover, in order to get meaningful clusters, CUTiS also merges a set of micro-groups. CUTiS decides if two micro-groups are candidates to be merged into a cluster if they are tangent or intersecting each other. The following definition describes more formally the concept of micro-groups that are merge candidates.

Definition 4.3.1 (**Micro-groups Merge Candidates**). Let the micro-group radius r_{g_i} be the distance between the representative trajectory and the farthest sub-trajectory of a moving object in g_i. For the sake of brevity, r_{g_k} is the micro-group radius of the micro-group g_k. Then, g_i and g_k are merge candidates if the following condition holds:

$$distance(traj[R_{g_i}^{traj}], traj[R_{g_k}^{traj}]) \leq r_{g_i} + r_{g_k}$$

.

By filtering out the merge candidates of a micro-group, CUTiS saves computation in the clustering process. The sub-trajectory clustering algorithm is composed of two phases. The first phase is to discover the merge candidates of a micro-group (explained before). In the second phase, CUTiS finally finds the clusters using the merge candidates by checking if two micro-groups candidates to merge are directly density reachable, and so density reachable. By merging micro-groups, the final clustering result is a density connected cluster which is approximated to the one produced by applying the original DBSCAN w.r.t. ε and τ.

Algorithm 6 illustrates how CUTiS finds the merge candidates of a micro-group (let's say, g_i). For a micro-group g_k (Line 2) whose representative trajectory is at a distance to the representative trajectory of the micro-group g_i (Line 3) not greater than the sum of their radius r_{g_i} and r_{g_k} (Line 4), then g_i and g_k are merge candidates (Line 5). Algorithm 6 returns the set of micro-groups which are merge candidates of the micro-group g_i.

Algorithm 6: Merge Candidates

Input:
- A micro-group at time window i, g_i.
- The set of micro-groups at time window i, G_i.

Output: The set of micro-groups candidates to be merged with g_i, MC.

```
1 MC ← ∅
2 foreach g_k ∈ {G_i \ {g_i}} do
3     d ← distance(traj[R^traj_{g_i}], traj[R^traj_{g_k}]);
4     if d ≤ r_{g_i} + r_{g_k} then
5         MC ← MC ∪ {g_k};
6 return MC;
```

Algorithm 7 first picks a micro-group and initialized it as a new cluster (Lines 4 - 5). For each micro-group g_i in a cluster, the algorithm checks its candidates to merge (Line 8), some micro-groups are filtered out. For each micro-group g_k candidate to merge with g_i, the algorithm checks if they are directly density reachable (Lines 9 - 13). If they are, the algorithm merges them into the same cluster(Lines 14 - 15). The algorithm outputs the density connected clusters when all micro-groups belonged to any cluster (Line 17).

4.4 Example of CUTiS approach

In this section we provide a running example of CUTiS. The first thing that CUTiS has to deal with is the creation (or maintenance) of micro-groups – Figure 7 shows a running

Algorithm 7: Sub-trajectory Clustering

Input:
- The set of micro-groups at time window i, G_i.
- Distance threshold ε.
- Size threshold τ.

Output: The set of clusters C.

```
1  begin
2      while G_i ≠ ∅ do
3          randomly pick a micro-group g_i from G_i;
4          Initialize cluster c ← g_i, add c to C;
5          remove g_i from G_i;
6          foreach g_i unvisited in c do
7              mark g_i as visited;
8              foreach g_k ∈ MergeCandidates(g_i, G_i) do
9                  foreach o_i ∈ g_i, o_k ∈ g_k do
10                     if distance(o_i, o_k) ≤ ε then
11                         n_i ← |N_ε(o_i)|;
12                         n_k ← |N_ε(o_k)|;
13                         if n_i ≥ τ and n_k ≥ τ then
14                             c ← c ∪ g_k;
15                             remove g_k from G_i;
16                             break;
17     return C;
```

example limited to this phase. In order to create a micro-group, CUTiS randomly picks up a moving object, and checks whether its sub-trajectory can be used as the representative trajectory of a new micro-group: in order to do so, CUTiS checks if the conditions specified in Definition 4.2.3 hold.

More precisely, let us suppose that we have $\varepsilon = 1$ meter, $\sigma = 2$ and $\rho = 0.17$ – as a consequence, the maximum radius a micro-group can have is ≈ 2 meters; let us finally suppose that $\tau = 3$. During the first time window two micro-groups are created – this corresponds to the *appear* pattern. Once we enter the second time window, the moving objects o_5 and o_{11} stop to send their positions and, as such, they must be *removed* from the system. Consequently, the deletion of o_5 entails the *splitting* of g_1 into two micro-groups, i.e., g_3 and g_4. This process is carried on by choosing new, suitable representative trajectories among the trajectories composing g_1 – this is done by means of the voting procedure. In the case of g_2 the micro-group *survives*, Finally, we notice that a new moving object, o_{16}, appears and starts to send its position during the second time window. CUTiS detects that o_{16} can be represented by the representative trajectory $R_{g_2}^{traj}$ of g_2 and, as such, includes the object in the micro-group.

At this point we show how CUTiS is able to infer clusters from micro-groups. Each micro-group is density-based: as such, micro-groups can be used to find density-based clusters

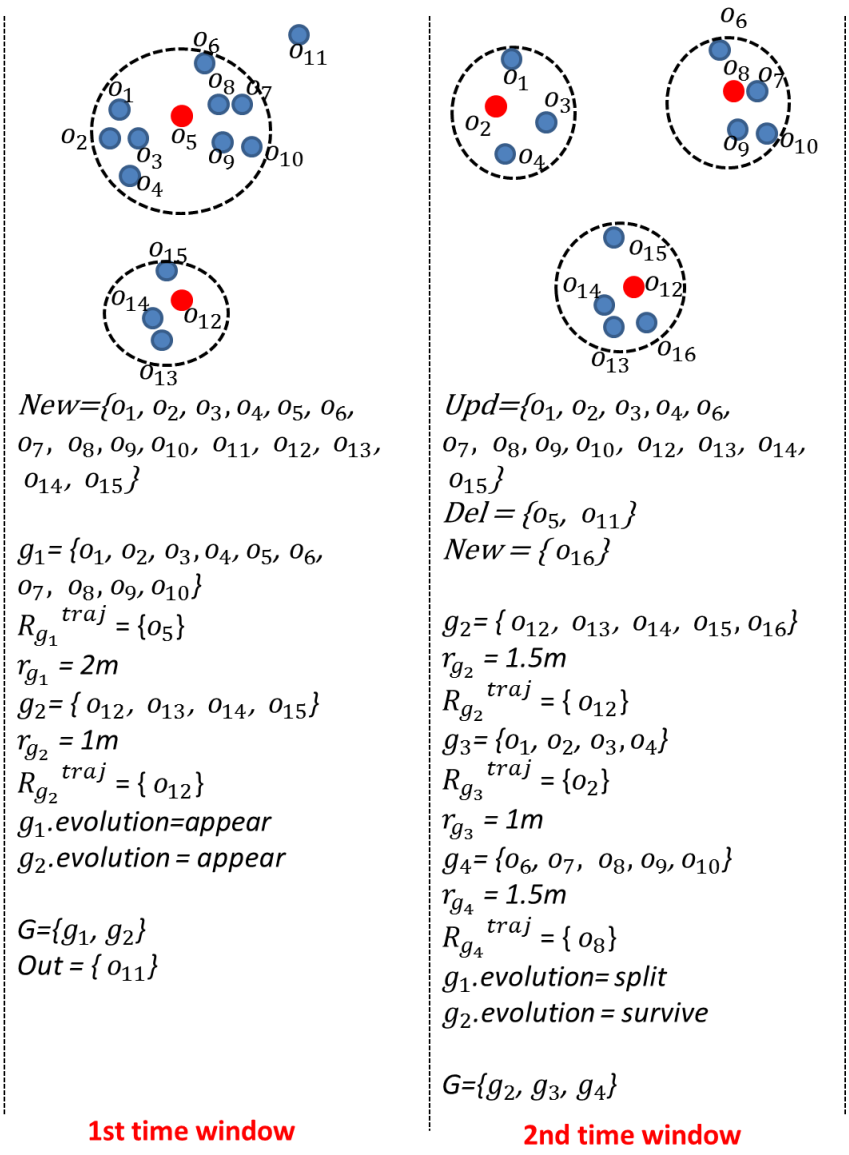


Figura 7 – Micro-Group Maintenance

of sub-trajectories, by merging two or more micro-groups. Figure 8 provides the example. We remind that our clustering algorithm is made up of two phases: first, it finds the merge candidates; secondly, it performs the actual density-based clustering. Going back to the example depicted in Figure 8(a), the algorithm finds two merge candidates, g_1 and g_2, and checks if $distance(traj[R_{g_1}^{traj}], traj[R_{g_2}^{traj}]) \leq r_{g_1} + r_{g_2}$ is satisfied. Since this holds, in the second phase the algorithm checks if g_1 and g_2 can be indeed merged, that is, if g_1 and g_2 direct density reachable (Figure 8(b)). Since we have that $o_1 \in g_1, o_2 \in g_2$, $distance(o_1, o_2) \leq \varepsilon$, $|N_\varepsilon(o_1)| \geq \tau$ and $|N_\varepsilon(o_2)| \geq \tau$, the algorithm proceeds to merge g_1 and g_2, thus creating a cluster.

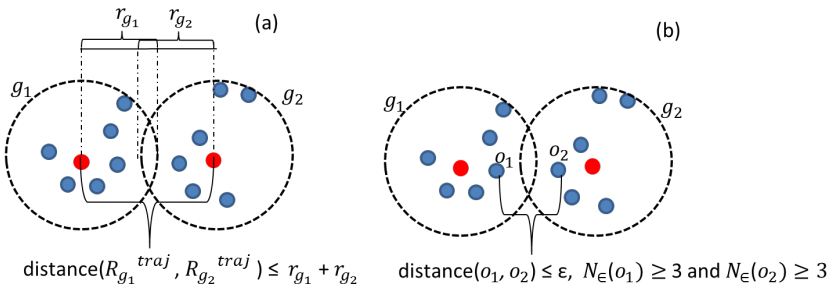


Figura 8 – Micro-Group Sub-trajectory Clustering

4.5 CUTiS*: Indexing Trajectory Data

4.5.1 Overview

In Section 4.2 we saw how CUTiS picks up moving objects randomly to create micro-groups – indeed, for each object CUTiS checks whether the associated sub-trajectory can be used as the representative trajectory of some micro-group. The drawback underlying this process is that the whole processing can get expensive, since each candidate sub-trajectory must be possibly compared against the whole dataset. As such, we aim to optimize this process by filtering out a relevant amount of objects during the creation of new micro-groups: to achieve this goal, we modify CUTiS to use spatial indices – from now on, we denote this variant by CUTiS*.

4.5.2 Grid construction and spatial indexing

Initially, CUTiS* builds a uniform grid G_i over the space in which moving objects move during the time window i.

Definition 4.5.1 (Space partitioning and indexing). Let $\mathscr{G}_i = (x_a^{\mathscr{G}_i}, x_b^{\mathscr{G}_i}, y_a^{\mathscr{G}_i}, y_b^{\mathscr{G}_i})$ be the minimum bounding rectangle (MBR) enclosing the space in which moving objects move during the time window i – as such, $(x_a^{\mathscr{G}_i}, y_a^{\mathscr{G}_i})$ and $(x_b^{\mathscr{G}_i}, y_b^{\mathscr{G}_i})$ represent, respectively, the lower-left and upper-right corners of the MBR; then, we partition $\mathscr{G}_i$ according to a uniform grid G_i of $N \cdot M$ cells of width W and height H such that the cell c_{ij} covers the following region:

$$(x_a^{\mathscr{G}_i} + i \cdot W, \quad x_a^{\mathscr{G}_i} + (i+1) \cdot W, \quad y_a^{\mathscr{G}_i} + j \cdot H, \quad y_a^{\mathscr{G}_i} + (j+1) \cdot H).$$

To ensure that the grid covers $\mathscr{G}_i$, constants N, M, W, and H are chosen so that $x_a^{\mathscr{G}_i} + N \cdot W \geq x_b^{\mathscr{G}_i}$ and $y_a^{\mathscr{G}_i} + M \cdot H \geq y_b^{\mathscr{G}_i}$ hold.

As such, each cell $C_{m,n}$ of G_i possibly encloses a subset of moving objects positions,

where m and n indicate the position of the cell in the grid. Let $o = (x,y)$ be the lower-left corner of the MBR $\mathscr{G}_i$; then, in order to find the cell $C_{m,n}$ in which a point $p = (x',y')$ falls, the following calculations suffice:

$$m = \lceil \frac{(x' - x)}{W} \rceil \qquad \text{and} \qquad n = \lceil \frac{(y' - y)}{H} \rceil \tag{4.9}$$

Once we have the tools to materialize a uniform grid over the space in which objects move, the next step is to describe how grids are indexed, such that spatial locality is preserved (up to some extent). We achieve this goal by means of some space-filling curve, i.e., functions that map a multidimensional space to a one-dimensional space such that two objects that are close in the original space are guaranteed to be close in the target space with some probability. As such, we leverage space filling curves to optimize range queries that involve sub-trajectories of moving objects. In our work we recur to Hilbert curves (JAGADISH, 1990) – we note that other types of space-filling curves can be used as well (e.g., Z-order). Consequently, the grid $\mathscr{G}_i$ gets covered by a Hilbert curve, while the index of each cell gets labeled with the related Hilbert index.

Finally, we report that CUTiS* stores the data related to the mapping of sub-trajectories (to Hilbert indices) in an ad-hoc data structure, and maintains a B-tree index over it by means of the attribute combination $(Hilbertindex, timestamp)$.

In the following we present an optimized algorithm that takes advantage of spatial indices to identify representative trajectories.

4.5.3 Selection of representative candidates

In order to find suitable representative trajectories, the candidate set is derived from the moving objects falling within dense grid cells, i.e., cells that contain more than τ moving objects. To take advantage of the fact that space-filling curves preserve spatial proximity (up to some extent), the candidate set also includes moving objects that fall in cells adjacent to dense ones. All in all, the usage of a spatial index allows CUTiS* to avoid having to consider the whole dataset when creating new micro-groups.

4.5.4 Range query computation

Range queries are computed in CUTiS during the initialization and maintenance phase of micro-groups, when retrieving the neighbors of representative trajectories. Range queries are computed during the execution of the sub-trajectory clustering algorithm as well, to discover whether two micro-groups are directly density connected.

In (SILVA *et al.*, 2016b) the authors proposed a distance function that considers time and space. Since each time window $i = [t, t+\delta t]$ gets divided into n intervals, each sub-trajectory of a moving object is augmented to signal its position at the boundaries of the intervals. In order to optimize the range query involving the sub-trajectory of a moving object o_j, CUTiS* queries the B-tree index (which, we remind, is based on the tuple {Hilbert Index, timestamp}) to retrieve moving objects that are "possible neighbors" of o_j. We sketch out in Algorithm 8.

Algorithm 8: Range Query

Input:
- Input stream at time interval $i = [t, t+\delta t]$, I_i.
- A moving object sub-trajectory $\{o_j, ST_{j,i}\}$.
- Distance threshold ε.

Output: A set of moving object, $N_\varepsilon^{o_j}$

```
1  begin
2      P ← ∅; N_ε^{o_j} ← ∅;
3      foreach (x,y,t) ∈ ST_{j,i} do
4      h_i ← getHilbertIndex(x,y);
5      P ← P ∪ queryBtree(h_i,t,I_i);
6
7      foreach {o_k, ST_{k,i}} ∈ P do
8      if distance(ST_{j,i}, ST_{k,i}) ≤ ε then
9          N_ε^{o_k} ← N_ε^{o_j} ∪ {o_k};
10     return N_ε^{o_j};
```

Algorithm 8 takes in input a moving object sub-trajectory $\{o_j, ST_{j,i}\}$ and returns its neighbors. First, there is a mapping between each position (x,y,t) of the sub-trajectory $ST_{j,i}$ (Line 3) and a hilbert index h_i (Line 4). Algorithm 8 performs a query over the B-tree index to find the moving objects having at least one position associated with the timestamp t and h_i. Some neighbors of o_j might fall in cells that are adjacent to h_i; as such, we retrieve object positions that fall within such cells (and are recorded at time instant t as well). This procedure is done in Line 5. For each "possible neighbor" $\{o_k, ST_{k,i}\}$, Algorithm 8 checks if $distance(ST_{j,i}, ST_{k,i}) \leq \varepsilon$ is true: if this holds, then $\{o_k\}$ is added to $N_\varepsilon^{o_j}$ (Line 9).

Even though the index proposed for CUTiS* is able to reduce the execution times,

some “core” moving objects were not labeled as core objects. Then, using the index in DBSCAN implementation for instance, can not guarantee the same result as the original DBSCAN. This fact happens because the index could not retrieve all moving object neighbors in the range query execution. Even if the index retrieves the moving objects which have sub-trajectory positions close to each other, it can not guarantee they will be close in the whole sub-trajectory shape. In this way, by using our proposed index in CUTiS or DBSCAN implementation, it loses accuracy compared to the original DBSCAN. We experimented this case and we present the results in the next section.

4.6 Experiments

To the best of our knowledge, there is no other incremental trajectory clustering algorithm for trajectory data stream. However, for validation purpose, we employ as a baseline some state-of-the-art methods: TraClus (LEE *et al.*, 2007) and DBSCAN, with the following changes: (1) at each time window we run DBSCAN and TraClus given as input only the sub-trajectories of the current time window, and not the complete trajectory; (2) we also adapted TraClus and DBSCAN to use our distance function, in order to take into account the distance in time, space and direction. We avoid to compare with (ESTER *et al.*, 1998) because to apply incremental clustering for each moving object trajectory at each time window is expensive (LI *et al.*, 2013).

4.6.1 Evaluation Metrics

The DBSCAN’s results are used as the baseline to test the effectiveness (recall and precision) because (1) DBSCAN has been regarded as the most representative density-based clustering algorithm and (2) both TraClus and CUTiS(or CUTiS*) share many characteristics with the algorithm DBSCAN. The effectiveness is measured according to the found clusters and outliers detection. In our case, the recall measure for outliers detection is the proportion of correctly classified as outliers (true positives) over DBSCAN’s outliers (true positives+false negatives). And the precision measure is the proportion of correctly classified as outliers (true positives) by CUTiS(or CUTiS*) over all the objects classified as outliers (true positives+false positives) by CUTiS (or CUTiS*). Let Out_i^{dbscan} and Out_i^{mg} be a set of moving object outliers found by DBSCAN and by CUTiS(or CUTiS*) on the time window i, respectively. The recall

and precision for outliers detection are measured as follows:

1. $Recall_{outlier} = \frac{|Out_i^{dbscan} \cap Out_i^{mg}|}{|Out_i^{dbscan}|}$
2. $Precision_{outlier} = \frac{|Out_i^{dbscan} \cap Out_i^{mg}|}{|Out_i^{mg}|}$

The recall and precision were also measured for the found clusters. At first, we matched each cluster C_i^{mg} found by CUTiS (or CUTiS*) with only one DBSCAN cluster C_i^{dbscan}. On the matching process, we applied Jaccard Similarity to compare the clusters based on their moving objects labeled as core (since the set of core moving objects is a deterministic result in DBSCAN). In this way, the cluster C_i^{mg} matches with one DBSCAN cluster (for example, C_i^{dbscan}) that presents maximum Jaccard similarity value. The recall and precision are applied for each matched pair to measure the quality of clusters result. The recall for $\{C_i^{mg}, C_i^{dbscan}\}$ is the proportion of correctly core moving objects classified by CUTiS (or CUTiS*) in C_i^{mg} (true positives) over the core moving objects classified by DBSCAN in C_i^{dbscan} (true positives+false negatives). The precision measure for $\{C_i^{mg}, C_i^{dbscan}\}$ is the proportion of correctly classified as core moving objects by CUTiS in C_i^{mg} (true positives) over all the core moving objects classified in C_i^{mg} by CUTiS (true positives+false positives). If two or more clusters of CUTiS match with the same DBSCAN cluster, we gather them in the same matching. We reported the average recall and precision for all the matched clusters. Let $Core_i^{dbscan}$ and $Core_i^{mg}$ be the set of core moving objects in C_i^{dbscan} and C_i^{mg} clusters, respectively.

1. $Recall_{\{C_i^{mg}, C_i^{dbscan}\}} = \frac{|Core_i^{dbscan} \cap Core_i^{mg}|}{|Core_i^{dbscan}|}$
2. $Precision_{\{C_i^{mg}, C_i^{dbscan}\}} = \frac{|Core_i^{dbscan} \cap Core_i^{mg}|}{|Core_i^{mg}|}$

We also measured the efficiency of CUTiS (and CUTiS*) against our competitors. We implemented CUTiS (and CUTiS*) and the baselines in Java, and tested them by using real data sets. The clustering parameters $eps = \varepsilon$ and $minPoints = \tau$ are the same for CUTiS, CUTiS*, DBSCAN and TraClus implementation. We set these parameters according to different data sets. In all experiments, we studied the influence of micro-group radius (and also the representativeness threshold ρ, they are related). The micro-group radius is varied according to the values eps, $2eps$ and $3eps$.

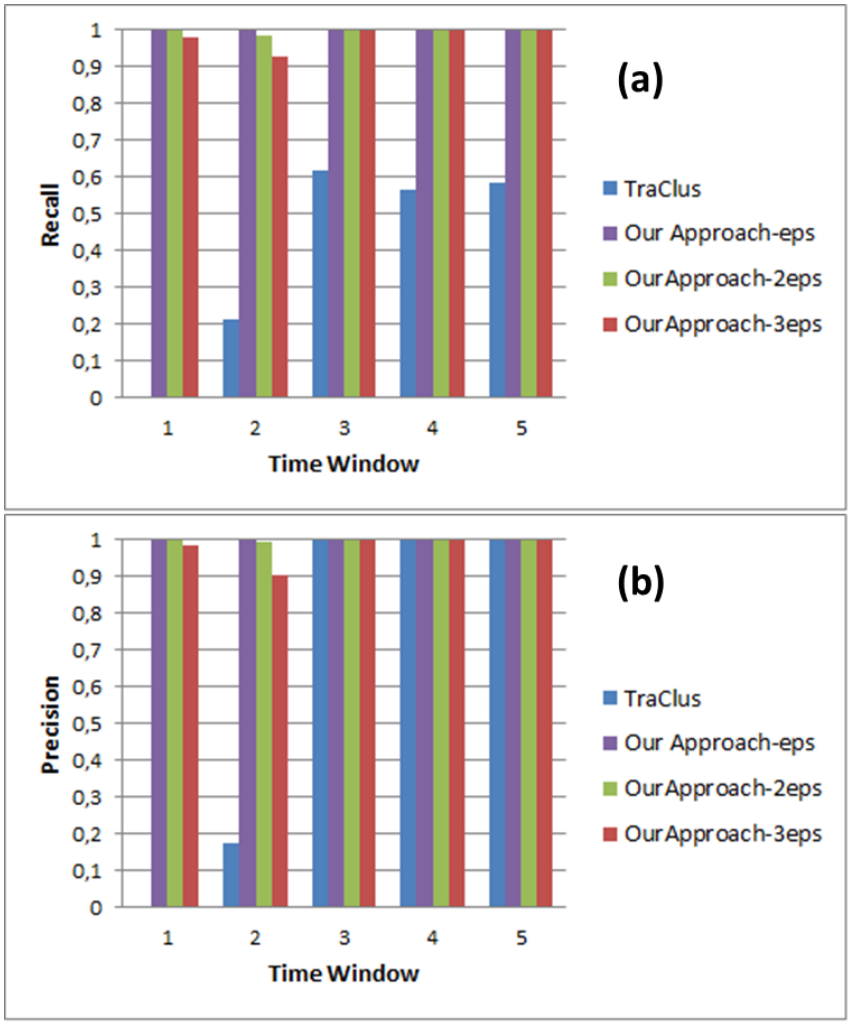


Figura 9 – Analysis of Taxi data set from Fortaleza: (a) Recall and (b) Precision for clusters detection for 5 time windows.

4.6.2 CUTiS Evaluation

4.6.2.1 Experiments Results for Taxi Fleet Dataset from Fortaleza

This dataset is computed from GPS recorded data of a taxi fleet in Fortaleza city, Brazil. It has around 400 different moving objects. Then, around 400 moving objects trajectories at each observed time window. It is a small dataset, but it is still dense because the sampling rate is equals to 1 second. The taxi fleet belongs to a private transportation company that tags and tracks its vehicles as they move around, and aims to support drivers' decisions related to displacement through the city in (quasi) real time (i.e., movement behaviors through the city that can be found by trajectory clusters analysis). In Figure 9, we evaluated the effectiveness to cluster sub-trajectories using CUTiS, DBSCAN and TraClus algorithm. We set the time window size as 5 (five) minutes and the clusters are tracked for 5 (five) sequential time windows. Figure

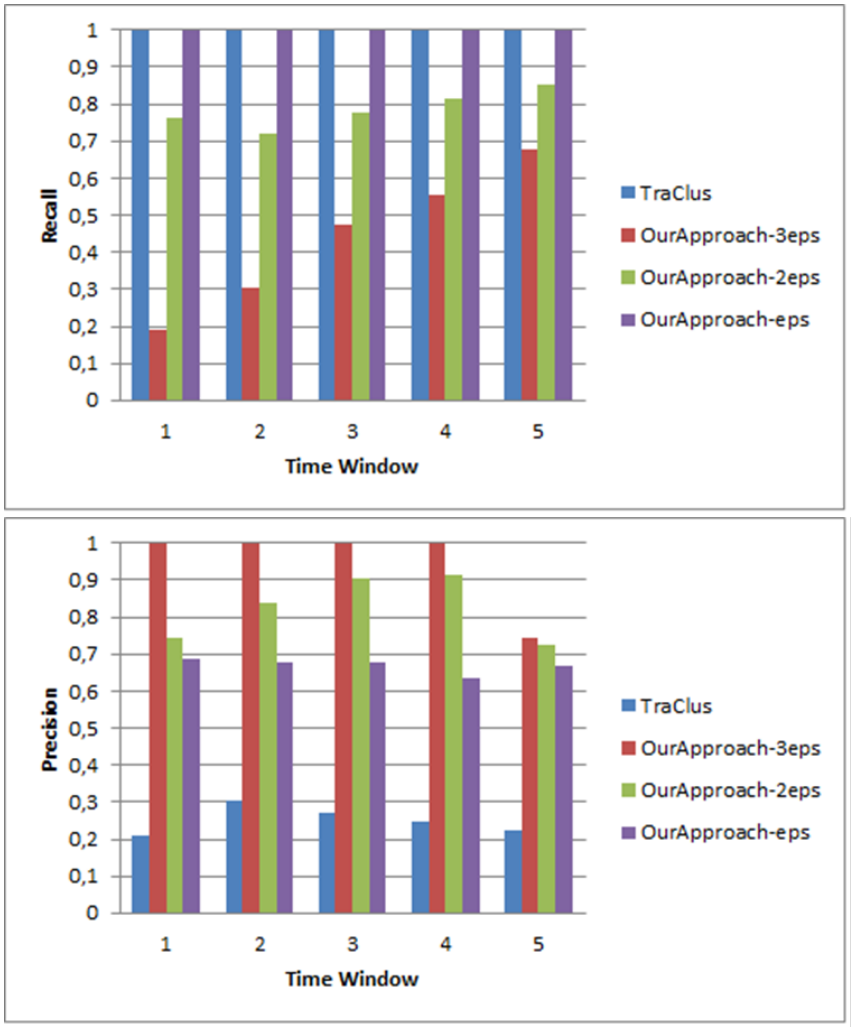


Figura 10 – Analysis of Taxi data set from Fortaleza: Recall and Precision for outliers detection for 5 time windows.

10 shows the effectiveness of outliers detection.

In Figure 11, note that CUTiS outperforms DBSCAN algorithm for all tested values of micro-group radius. When (1) micro-group radius=*eps*, CUTiS is from 20% to 54% faster than DBSCAN, for (2) radius=2*eps*, it is from 70% to 90% faster and for (3) radius=3*eps*, it is from 88% to 92% faster than DBSCAN. When micro-group radius increases, CUTiS also outperforms TraClus, because in this case there are less micro-group to be maintained. Even if TraClus outperformed CUTiS when radius=*eps*, the obtained clusters do not match with those of DBSCAN (our baseline).

As presented in Figure 9, we can see the quality of our clusters is higher than TraClus. CUTiS may misclassify outliers, and the recall for outliers detection decreases (Figure 10(a)). As expected, when the micro-group radius=*eps*, CUTiS does not misclassify outliers (since each micro-group only contains the representative trajectory which is a core object according to

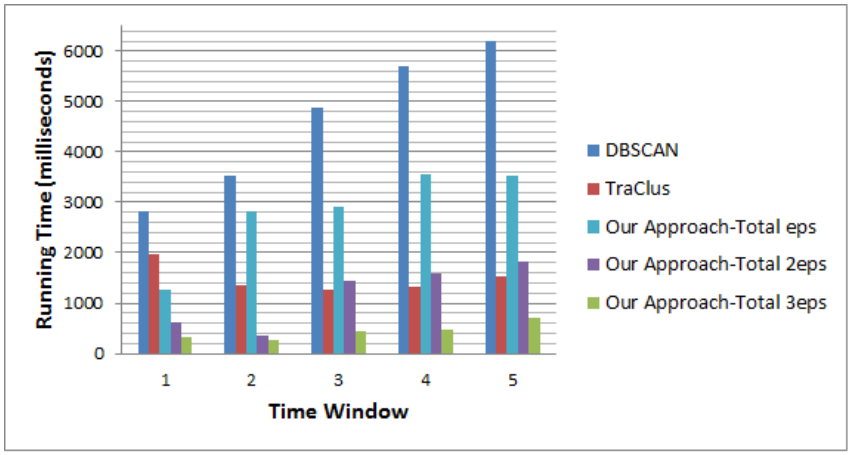


Figura 11 – Analysis of Efficiency on Taxi data set from Fortaleza

DBSCAN definition, and its directly density reachable sub-trajectories). However, there are some false positive outliers as well as in TraClus result. This affects the precision measure showed in Figure 10(b). This trade-off between precision and recall in outlier detection depends on the application needs. However we believe that CUTiS is quite satisfactory for both measures since it does not present very low values. In the next section, we will discuss in detail the experiments on a much larger number of moving objects, for better illustration of the advantages of CUTiS.

4.6.2.2 Experiments Results for Taxi Data Set from Beijing

This data set contains a real world GPS recorded data from taxis of Beijing (called T-DRIVE dataset) (YUAN *et al.*, 2011; YUAN *et al.*, 2010). We have used a trajectory subset of this data set containing around 7,000 moving objects, then 7,000 trajectories referring to taxi travels around Beijing city at each time window. In the experiments, we set the time window size as 10 (ten) minutes and the clusters are tracked for 5 (five) sequential time windows. The sampling rate for T-DRIVE is around 1 minute. We chose to experiment as well with T-DRIVE (not only the Taxi Fleet dataset from Fortaleza, Brazil) to verify the efficiency of our approach for a large dataset of moving objects.

Throughout these experiments, we set the CUTiS parameters for the clustering algorithm equals to $\varepsilon = 0.4$ and $\tau = 50$.

Effectiveness Analysis of the Clustering approach. The effectiveness analysis is measured by applying the precision and recall metrics introduced above, and compares the shapes of the clusters in DBSCAN, TraClus and CUTiS where the clustering is computed by merging micro-groups. We expected that our method finds more clusters than DBSCAN, given the fact that the merge of two or more micro-groups only compares their representatives trajectories (this

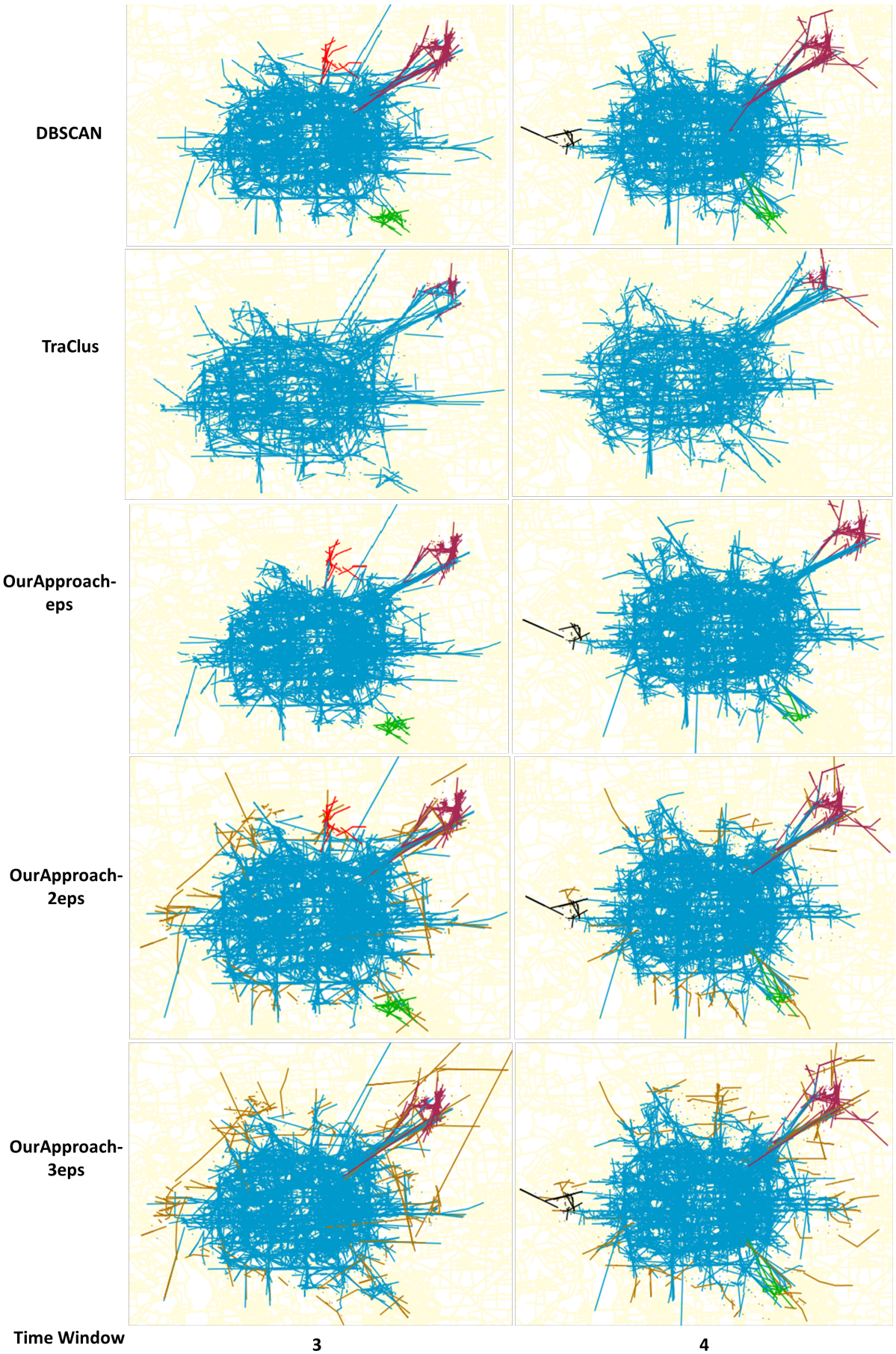


Figura 12 – Comparison between clusters shapes of DBSCAN, TraClus and CUTiS (varying the micro-group radius) for 2 time windows using Taxi data set from Beijing

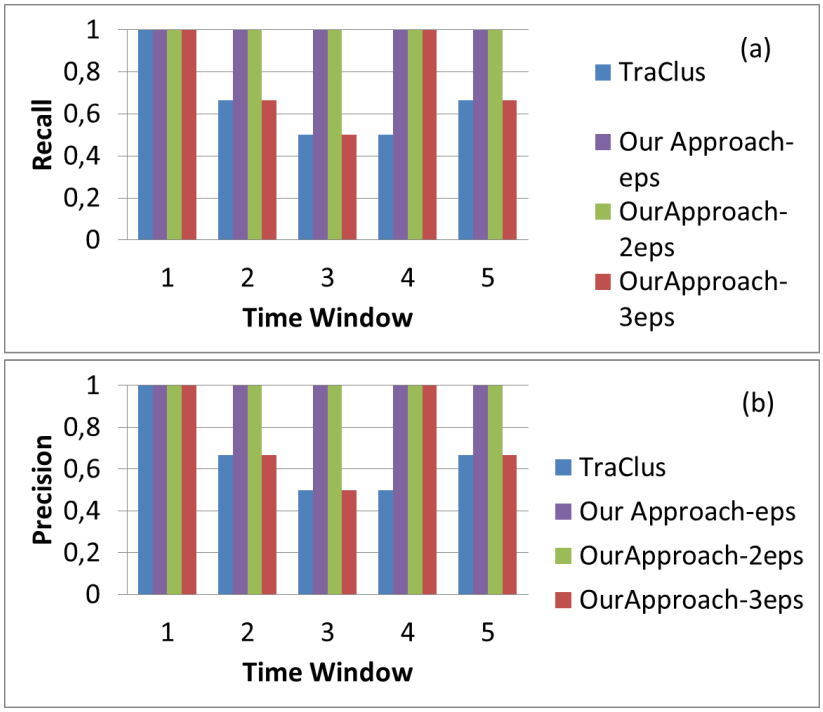


Figura 13 – Effectiveness Analysis: (a) Recall and (b) Precision for detecting clusters using Taxi data set from Beijing

allows us to save the cost of pairwise sub-trajectory distance computation in DBSCAN). So each DBSCAN cluster is expected to match with many clusters from CUTiS. Actually, in our experiments, CUTiS found the same clusters as DBSCAN when the micro-group radius=eps and micro-group radius=2eps (except very few differences). Figure 12 compares the shapes of the clusters in DBSCAN, TraClus and CUTiS. Figure concentrates in only two time windows and each color (blue, red, purple, green and black) represents one DBSCAN cluster. We obtain the best matches with DBSCAN, in both time windows, especially with radius=eps.

The differences are greater when micro-groups have radius=3eps. In the third time window, CUTiS included the clusters represented by green and red colors into the same cluster (represented by blue color). This means that a cluster found by CUTiS contains core objects from three different DBSCAN clusters. This happens because these core objects can be represented by their micro-group representative trajectory, however the core objects are not density reachable objects (according to DBSCAN definition) to be in the same cluster. TraClus did not produce similar clusters to DBSCAN for both time windows, the clusters represented by red, green and black colors were not found. Furthermore, there are many moving objects in DBSCAN clusters that were not clustered by TraClus. The main reason for the difference between TraClus and DBSCAN results is the preprocessing phase in TraClus which partitions the trajectories which

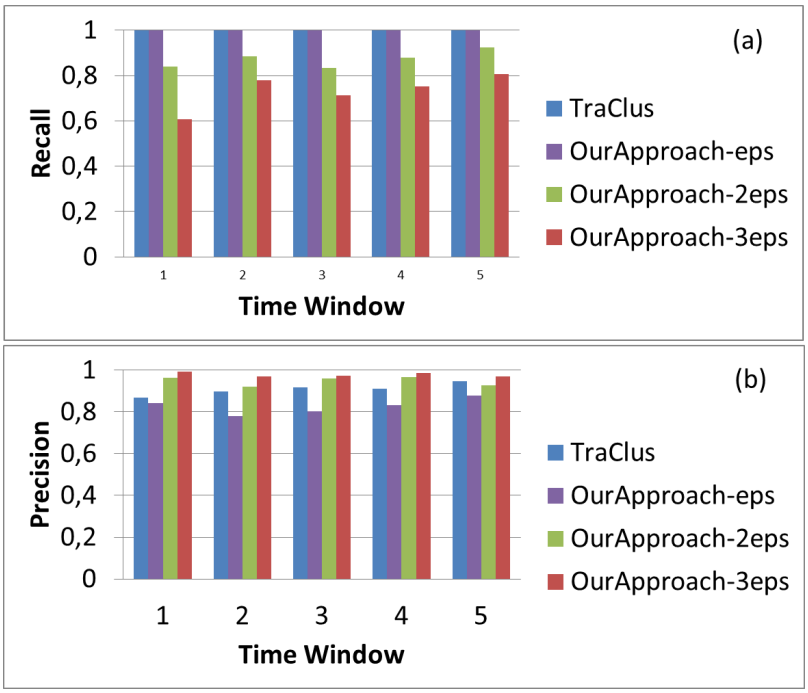


Figura 14 – Effectiveness Analysis: (a) Recall and (b) Precision for detecting outliers using Taxi data set from Beijing

modifies the similarity since it is applied per partition instead of the whole sub-trajectories.

As expected, CUTiS presents higher precision and recall for cluster detection when the micro-group radius=eps and micro-group radius=$2eps$ (Figure 13), since the clusters produced by CUTiS, in these cases, have similar shapes to DBSCAN clusters. Unlike when the micro-group radius=$3eps$ and for TraClus approach (as we previously discussed). In Figure 12, we highlighted with brown color the misclassified outliers by CUTiS. When micro-group radius becomes larger, it means that the representativeness threshold is higher, then some outlier sub-trajectories have enough votes to be reached by one micro-group representative trajectory. This holds for radius=$2eps$ and micro-group radius=$3eps$. In this case, CUTiS may misclassify outliers, and the recall for outliers detection decreases (Figure 14(a)). As expected, when the micro-group radius=eps, CUTiS does not misclassify outliers (since each micro-group only contains the representative trajectory which is a core object according to DBSCAN definition, and its directly density reachable sub-trajectories). However, there are some false positive outliers as well as in TraClus result. This affects the precision measure showed in Figure 14(b). So when micro-group radius increases, CUTiS may misclassify outliers but also brings down the number of false positive outliers. This means the recall decreases and the precision increases for outliers detection. This trade-off between precision and recall in outlier detection depends on the

application needs. However we believe that CUTiS is quite satisfactory for both measures since it does not present very low values.

Efficiency Analysis of the Clustering approaches. Data stream applications impose a limited memory constrained, it becomes difficult to provide arbitrary-shaped clustering results using conventional algorithms as DBSCAN. It is unrealistic to provide such a precise result. In the original DBSCAN, the system has to check the density connectivity for each moving object sub-trajectory which makes the approach computational costly (showed in Figure 15(a)). To discover density based clusters from trajectory data stream, CUTiS has two steps namely the creation and maintenance step (Mstep, Algorithms 3, 4 and 5), and clustering step (Cstep, Algorithm 6). In Figure 15(b), the clustering step dominates the total cost in most of the time windows. Since to merge micro-groups into a cluster, we need to detect if they are density connected which is expensive (in the worst case, it is necessary to compare all pairwise moving objects that belong to these micro-groups). However, the micro-group structure can save computation in the maintenance processing avoid accessing all data set objects in details. Our results present more gain in efficiency than DBSCAN when micro-group radius increases (it maintains less representative trajectories, i.e. less number of micro-groups). As we mentioned before, TraClus approximated in the experiments most of the time the sub-trajectory to only one segment. In this case, the distance computation is less costly than consider each sub-trajectory with many segments as CUTiS did. This is the main reason why TraClus is much faster to do clustering. However, it only essentially outperformed CUTiS when micro-group radius=eps.

In general, our results present a trade-off between quality and performance, it is influenced by the micro-group radius value. Generally speaking, when the radius has low value, i.e. the micro-groups size decreases, it leads to maintain too many representatives (low performance) but our clustering algorithm produces similar result to DBSCAN (high quality). By increasing the micro-group radius, CUTiS presents high performance but it may lead to misclassify outliers (low quality).

Micro-group Evolution Pattern. CUTiS captures the micro-group evolution. Figure 16 reports the percentage of micro-groups that evolve according to each pattern for all the tracking time windows. One can clearly see that the proportion of unchanged micro-groups (micro-groups that survive) is greater when micro-group radius is lower. The intuition is when the radius is lower, there are more micro-groups, and consequently it is necessary to choose more representatives. The probability to randomly choose good ones is greater than when micro-group

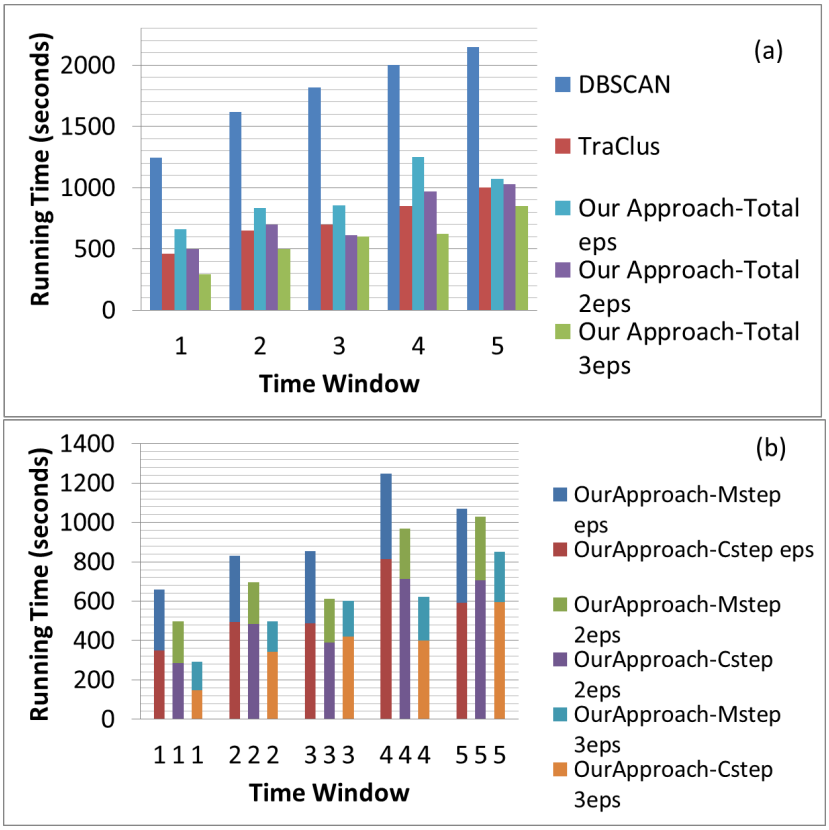


Figura 15 – Efficiency Analysis: (a) Total Running Time for 5 time windows using Taxi data set from Beijing and (b) Running Time of Micro-group maintenance and clustering

radius increases (less representatives trajectories are chosen). A micro-group splits when its representative trajectory does not have enough votes to remain the representative in the current time window. There is less micro-groups that split when radius=*eps* (CUTiS got more "good" ones representatives) than when micro-group radius=3*eps*.

Hence we suggest that in real applications, the user should set higher micro-group radius to achieve higher performance to cluster sub-trajectories. However, if the priority is to have quality in the results (considering DBSCAN as the baseline), we suggest to set lower value for micro-group radius in order to filter out outliers, and to guarantee the algorithm's sensitivity.

4.6.3 CUTiS Evaluation*

The experiments use a subset of a real world GPS recorded data from taxis of Beijing, containing around 4,000 trajectories. Two possible scenarios were investigated which were not investigated before. When a moving object stops sending its positions: (1) **Scenario I**: it should be deleted in the next time window or (2) **Scenario II**: its sub-trajectory positions should be predicted until a timeout is reached (here the timeout is equals to 2 time windows, i.e., 14

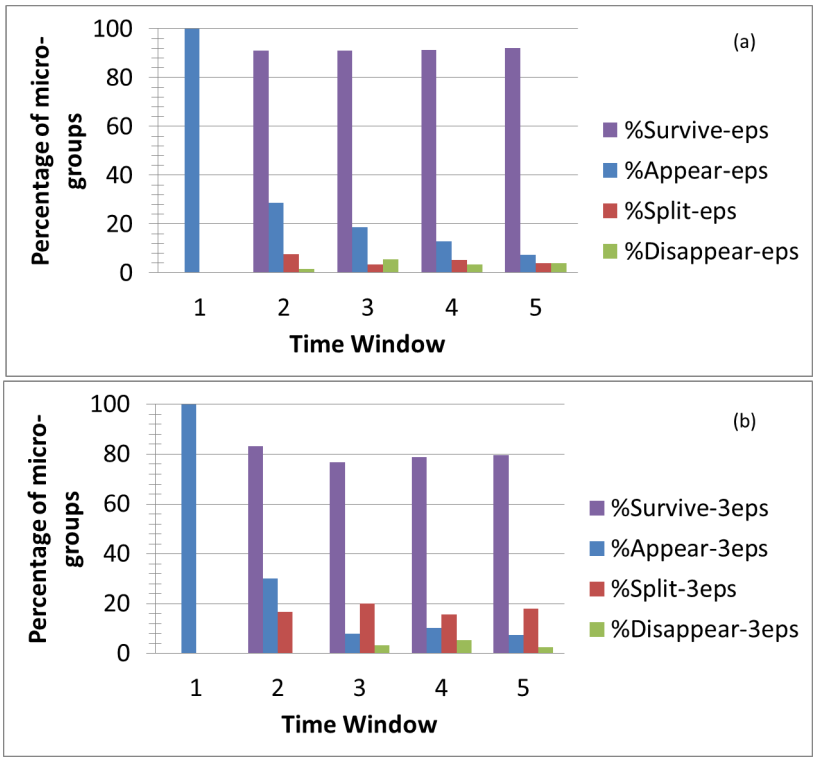


Figura 16 – Micro-group evolution pattern for (a) radius=eps and (b) radius=3eps on the 5 tracked time windows using Taxi data from Beijing

Tabela 7 – Number of Clusters

Time Window	1	2	3	4	5
Scenario I	14	10	12	15	14
Scenario II	14	15	22	16	15

minutes). Both scenarios use a time window size of 7 (seven) minutes and the clusters are tracked for 5 (five) sequential time windows. The number of clusters on the 5 tracked time windows is shown in Table 7 by using CUTiS* with micro-group radius equals to *eps*.

Throughout these experiments, we set the CUTiS* parameters for the clustering algorithm equals to $\varepsilon = 0.004$ and $\tau = 20$.

Micro-group Evolution Pattern. CUTiS* captures the micro-group evolution. For both scenarios, Figure 17 reports the percentage of micro-groups that evolve according to each pattern for all the tracking time windows. Each scenario has its peculiarities which affect the micro-groups maintained. In Scenario I, there exist moving objects deleted at each time window; hence the proportion of micro-groups that disappear or split is greater than in Scenario II. However, even if the majority of micro-groups survive and there exist moving objects whose positions are predicted, Scenario II is not far from the real-world objects. Between 30% to 40% of micro-group split in Scenario II.

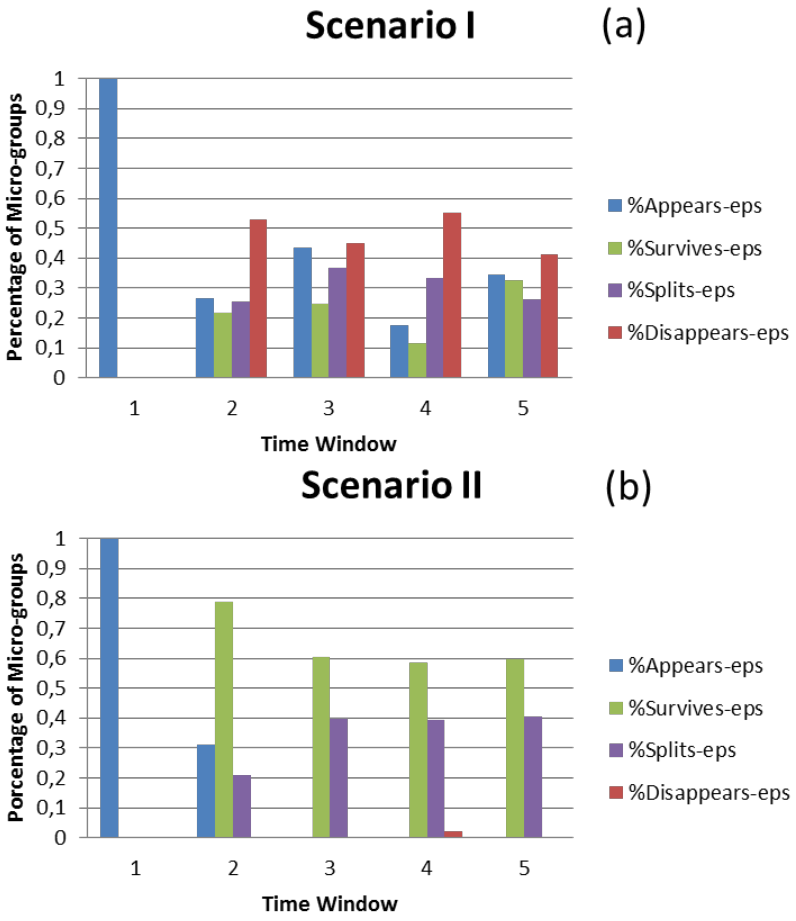


Figura 17 – Micro-group evolution pattern found by CUTiS* during 5 tracked time windows.

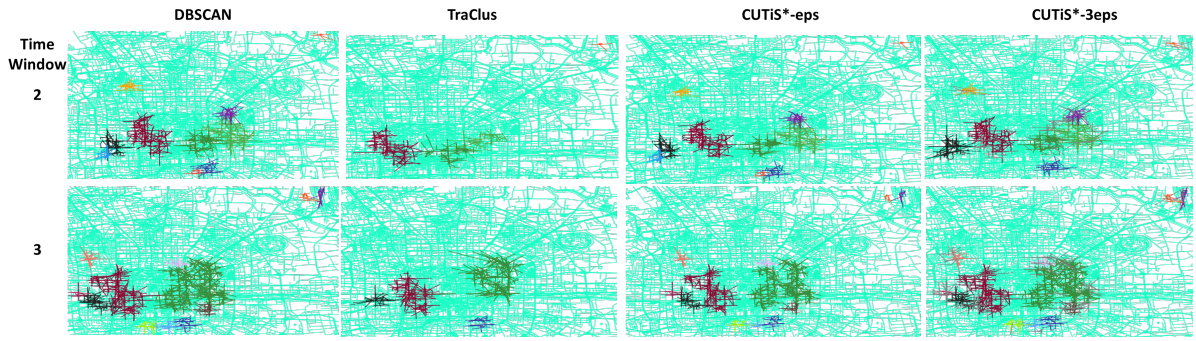
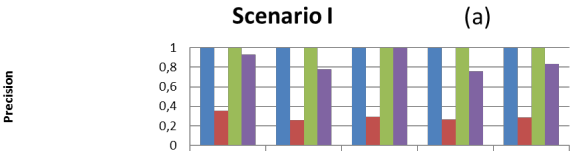


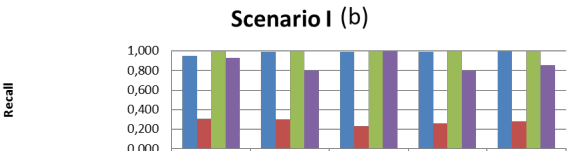
Figura 18 – Comparison between clusters shapes of DBSCAN, TraClus and CUTiS* (varying the micro-group radius) for 2 time windows using data of Scenario I.

Effectiveness Analysis. The effectiveness analysis is measured by applying the precision and recall metrics and comparing the clusters in DBSCAN (the baseline), DBSCAN with index, TraClus and CUTiS*. In our experiments, CUTiS* found the same clusters as DBSCAN when the micro-group radius=*eps*. Figure 18 compares the shapes of the clusters in DBSCAN, TraClus and CUTiS*. Figure 18 concentrates in only two time windows and each color represents one DBSCAN cluster. We obtain the best matches with DBSCAN, in both time

windows, especially with radius=*eps*. The clusters found by DBSCAN with index are omitted in Figure 18 because they are quite similar to DBSCAN clusters except few differences as we will see later, some cores in DBSCAN are not label as core in DBSCAN with index.



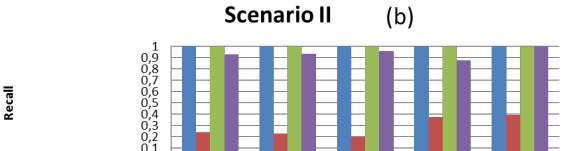
	1	2	3	4	5
Precision DBSCAN-with Index	1	1	1	1	1
Precision Traclus	0,357	0,260	0,293	0,266	0,286
Precision CUTiS*-eps	1	1	1	1	1
Precision CUTiS*-3eps	0,928	0,777	1	0,758	0,831



	1	2	3	4	5
Recall DBSCAN-with Index	0,947	0,994	0,991	0,988	0,997
Recall Traclus	0,310	0,298	0,233	0,256	0,279
Recall CUTiS*-eps	1	1	1	1	1
Recall CUTiS*-3eps	0,929	0,800	1	0,800	0,857

Figura 19 – Effectiveness Analysis Scenario I: (a) Precision and (b) Recall for detecting clusters

	1	2	3	4	5
Precision DBSCAN-with Index	1	1	1	1	1
Precision Traclus	0,286	0,267	0,225	0,375	0,392
Precision CUTiS*-eps	1	1	1	1	1
Precision CUTiS*-3eps	0,928	0,933	0,936	0,844	1



	1	2	3	4	5
Recall DBSCAN-with Index	1	1	0,999	1	1
Recall Traclus	0,239	0,229	0,198	0,372	0,389
Recall CUTiS*-eps	1	1	1	1	1
Recall CUTiS*-3eps	0,929	0,933	0,955	0,875	1

Figura 20 – Effectiveness Analysis Scenario II: (a) Precision and (b) Recall for detecting clusters

When micro-groups have radius=3*eps*, the differences are greater. In the second time window, CUTiS* included the clusters represented by black and light blue colors into the same cluster (represented by black color). The clusters represented by red and dark blue colors are also included into the same cluster (represented by dark blue color). This means that a cluster found by CUTiS* contains core objects from two different DBSCAN clusters. Indeed, these core objects can be represented by their micro-group representative trajectory, however the core objects are not density reachable objects (according to DBSCAN definition) to belong to the same cluster.

As expected, CUTiS* presents higher precision and recall for cluster detection when the micro-group radius=*eps* (Figure 19 and 20), since the clusters produced by CUTiS*, in these cases, have similar shapes to DBSCAN clusters. Unlike for TraClus and when the micro-group has radius=3*eps*, both have in all tracked time windows lower precision and recall values. TraClus is based on DBSCAN algorithm (LEE *et al.*, 2007). However, TraClus did not produce similar clusters to DBSCAN for all the tracked time windows. Furthermore, there are many moving objects in DBSCAN clusters that were not clustered by TraClus. We can clearly see that

some clusters were not found in the second time window (e.g., black, light blue, dark blue colors) in Figure 18. The main difference between TraClus and DBSCAN results is the preprocessing phase in TraClus which partitions the trajectories, then it modifies the similarity since it is applied per partition instead of the whole sub-trajectories. So even the prepocessing phase in TraClus increases its performance, the effectiveness of CUTiS* outperforms it.

Even though our proposed index accelerates the performance for DBSCAN (as we will see), some "core" moving objects were not labelled as core objects. Then, using the index in DBSCAN implementation can not guarantee the same result as the original DBSCAN. This fact happens because the index could not retrieve all moving object neighbors in the range query execution. Even if the index retrieves the moving objects which have sub-trajectory positions close to each other, it can not guarantee they will be close in the whole sub-trajectory shape. In this way, by using our proposed index in DBSCAN implementation, it loses accuracy compared to the original DBSCAN.

Efficiency Analysis. Figure 21(a) and (b) compare the running time of DBSCAN, our approach with micro-group radius=eps without the index for trajectory data (called "CUTiS-eps-without index") and our approach for micro-group radius=eps with our proposed index (called "CUTiS*-eps-with Index"). Our goal here is to show the improvement of CUTiS* due to the proposed indexing scheme. As expected, the performance of our approach using the index is much better than without index. By using the index, the speed-up of CUTiS* compared to CUTiS (without index) ranges from 2.5 to 5. In general, CUTiS* spent 10% of the running time to create the index at each time window (from 30 to 40 seconds). CUTiS* outperforms DBSCAN, since DBSCAN requires to check the density connectivity for each moving object sub-trajectory with the whole dataset, which is computationally costly.

To faithfully compare, we adapted the DBSCAN implementation to use the same index as CUTiS*. Figure 21(c) and (d) show the running time for DBSCAN with index, TraClus and CUTiS* with micro-group radius=eps and micro-group radius=3eps. As expected, the index also improved the DBSCAN performance. Even though the running time difference between CUTiS* and "DBSCAN-with index" is not large, DBSCAN can not track the evolution of clusters/groups as CUTiS* does and the clusters found by
"DBSCAN-with index" are not the same as those of the original DBSCAN (as we discussed before, "DBSCAN-with index" lost accuracy compared to the original DBSCAN). Our results present more efficiency gains than "DBSCAN-with index" when the micro-group radius increases

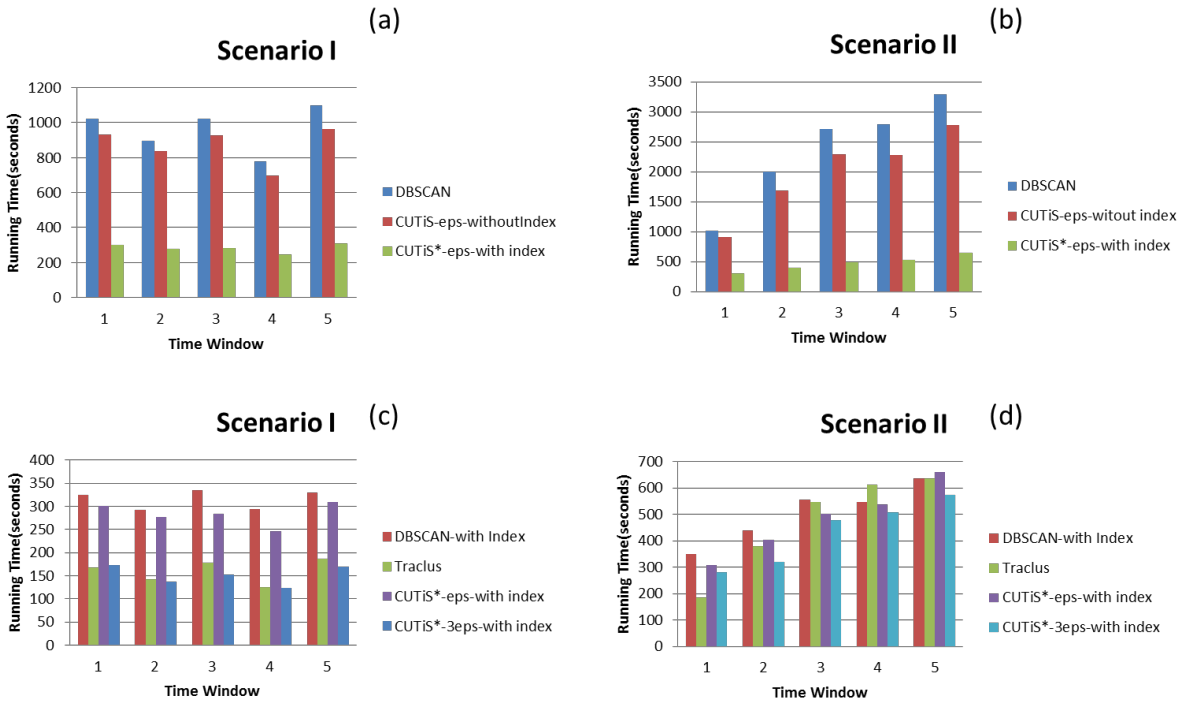


Figura 21 – (a) and (b): Efficiency Analysis (Improved Running Time by using our proposed index); (c) and (d): Efficiency Analysis of Clustering Approaches

(it maintains less representative trajectories, i.e., a smaller number of micro-groups). TraClus approximated in the experiments most of the time the sub-trajectory to few segments, as we mentioned before. In this case, the distance computation is less costly than considering each sub-trajectory with many segments, as CUTiS* did. The segmentation reduces the computational complexity and this is the main reason why TraClus is much faster to create the clusters.

In general, CUTiS* presented a trade-off between quality and performance, which is influenced by the micro-group radius value. Generally speaking, when the radius has a low value, i.e., the micro-groups size decreases, it leads to maintaining too many representatives (low performance), but our clustering algorithm produces similar result to DBSCAN (high quality). By increasing the micro-group radius, CUTiS* presents high performance but it may lead to misclassify outliers (reducing the quality). Moreover, by using our proposed index, CUTiS* improved its performance when compared with CUTiS (i.e., without index).

4.7 Discussion

In this chapter, we comprehensively discussed CUTiS, our framework to find mobility patterns on trajectory data stream. CUTiS addresses the problem of discover and maintain sub-trajectory clusters at each time window, since each moving object updates its sub-trajectory from

time to time, new moving objects appear and others disappear from the system. As each moving object evolves in time, also cluster evolves. It is inefficient as we showed in our experiments to compute trajectory clusters from scratch. There is a need to online track trajectory clusters over sliding window. This problem is also demanded in trajectory clustering applications as monitoring hot traffic paths, analyzing people's movement and understanding city dynamics.

Our framework CUTiS is composed by four steps: apply a distance function, choose representative trajectories, maintain the micro-groups and discover sub-trajectory clusters. Throughout this chapter, we presented our approach to track moving objects by applying the Synchronous Euclidean Distance. We define a new structure called micro-group, to capture and maintain small and dense groups of moving objects around the representatives at each time window. This adapts to many situations such as families staying together or persons sharing the same trip using a public transport. Furthermore, their maintenance cost is lower than maintaining a big cluster during each time window. CUTiS incrementally maintains each micro-group and captures its evolution patterns from time to time. By merging micro-groups, CUTiS discovers density based sub-trajectory clusters. Our experiments were conducted on real data sets from Taxis of Beijing and Taxis in Fortaleza city, and it shows the efficiency and effectiveness of CUTiS compared to our competitors. These results were published in (SILVA *et al.*, 2016b; SILVA *et al.*, 2016a).

We also proposed a sub-trajectory index to accelerate the sub-trajectory clustering processing in CUTiS. Our proposed index is based on a space-filling curve which has the property of mapping a multidimensional space to one-dimensional space such that, for two objects that are close in the original space, there is a high probability that they will be close in the mapped target space. We take advantage of this property to optimize range queries from a moving object sub-trajectory on the incremental clustering algorithm and develop CUTiS*. The proposed index improves the performance without losing quality in the clusters results as we show in our experiments conducted on a real dataset of Taxis of Beijing. These results were published in (SILVA *et al.*, 2016c).

We proposed a demo tool which was published in (SILVA *et al.*, 2016d) [1] and it will be integrated to Graphast framework in future (MAGALHÃES *et al.*, 2015). Our demo implements CUTiS* for online mobility pattern discovery in sub-trajectory data streams. Key innovations include: (1) Online discovery of mobility patterns and pattern evolution by tracking

[1] https://github.com/ARiDa/cutis

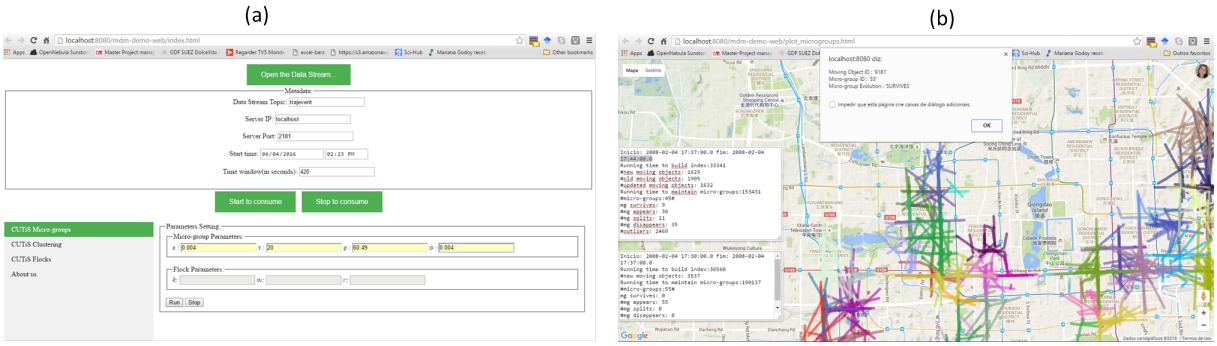


Figura 22 – Demonstration Scenarios.

the sub-trajectories of moving objects; (2) Micro-group trajectory discovery, density-based trajectory clusters and trajectory flocks discovery are available functionalities in our demo; (3) An incremental algorithm to maintain those patterns and capture their evolution on highly dynamic sub-trajectory data. Demonstration scenarios are showed in Figure 22 using a real dataset.

5 NET-CUTIS: CLUSTERING ON TRAJECTORY DATA STREAM ON ROAD NETWORK

In the previous chapters we investigated the problem of finding clusters of sub-trajectories in a 2D Euclidean space, as well as how to update and maintain clusters over time. In this context, we remember that we employ a distance function that takes as input the sub-trajectories of two moving objects and determines how similar their movements are. However, such function does not take into account the fact that, in real-world applications, movements of moving objects are typically constrained by the road network. As such, in this chapter, we explore the problem of sub-trajectory clustering on road networks which corresponds to **Challenge 4** of this thesis.

Before delving into the problem, we need to highlight several key differences that stand between an *Euclidean* space and a *Road Network* space:

- The distance between two objects on a road network is equivalent to the *length* of the *shortest path* connecting them, while in Euclidean spaces this is simply given by the length of the segment connecting the objects.
- While the (x,y,t) coordinate system is the most popular used with Euclidean spaces, (Sid,d,t) represents a more efficient coordinate system for road network spaces (HWANG *et al.*, 2005): here, *Sid* represents a *road sector identifier*, d represents an *offset* from the *starting point* of the road sector, and t represents a time instant associated with the object coordinate. Another, alternative representation (TIAKAS *et al.*, 2006; TIAKAS *et al.*, 2009) omits d: in this thesis, we adopt the latter.
- Road networks require additional data to describe the connectivity between road sectors.

Network connectivity is usually modeled by means of a graph representation, which is composed of a set of vertices (nodes) and a set of edges (connections). Depending on the application, the graph may be weighted (each edge has associated a cost) or directed (each edge has an orientation). In this work, we represent road networks as *weighted* (the cost associated with an edge corresponds to its length) and *directed* graphs.

In this context we assume that spatial coordinates can be matched efficiently to the edges of a road network. Several algorithms solve efficiently this problem (we refer the reader to (ZHENG, 2015) for an overview); however, developing an efficient map matching algorithm is out of the scope of this thesis, hence in this thesis we recur to a naive approach. More precisely, the map matching algorithm associates each position of a moving object, (x,y), with the nearest

edge (road sector), i.e. the edge having the minimum perpendicular distance.

Building on the CUTiS framework, we need to define a distance function capable to compare sub-trajectories in a road network space. Subsequently, in order to discover sub-trajectory clusters we first need to choose a representative trajectory and maintain the micro-groups.

In the following section (Section 5.1) we present two contributions: first, we introduce a spatio-temporal distance function that is suitable for sub-trajectory data constrained by a road network. Second, on the basis of the aforementioned contribution, we introduce an incremental clustering approach that is able to *detect the clusters online and their evolutions*.

5.1 Distance Function on Road Network

Let a road network be represented by a graph $G = (V, E)$, where V is the set of nodes and E the set of edges (road segments). The trajectory TR_j describes the movements of a moving object o_j, it is defined as

$$TR_j = ((e_1, (t_1, t_2)), (e_2, (t_2, t_3)), ..., (e_n, (t_n, t_{n+1}))), \tag{5.1}$$

where n is the sub-trajectory description length, e_i represents an edge in $G(V,E)$, and (t_i, t_{i+1}) represents the time interval during which o_j traverses e_i (o_j enters at time t_i and leaves at time t_{i+1}).

The spatio-temporal distance function proposed in this thesis depends on the concept of *network path*: intuitively, a network path represents the longest *sequence* of *road segments* that are shared among many trajectories.

More formally we have the following:

Definition 5.1.1. (**Network Path**) Consider a set of moving object trajectories $T = \{(o_1, TR_1), (o_2, TR_2), .., (o_m, TR_m)\}$. Then, $np_i = ((e_1, e_2, ..., e_k))$ is a *network path* of T if and only if:

- $\forall j \in \{2, ..., k-1\}$, $e_{j-1}.end = e_j.start$
- Let $L = \{TR_1, ..., TR_p\}$ be a non empty set: then $\forall j \in \{1, ..., k\}, \forall q \in \{1, ..., p\}$, $\exists t_u, t_w$, with $t_u < t_w$, such that $(e_j, (t_u, t_w)) \in TR_q$.

Consider a sub-trajectory $ST_{j,i} = \{(e_1, (t_1, t_2)), (e_2, (t_2, t_3)), ..., (e_n, (t_n, t_{n+1}))\}$ related to a moving object o_j at a time window i. Consider also a set of network paths

$NP = \{np_1, ..., np_m\}$. In order to "summarize" $ST_{j,i}$ using NP, we exploit the algorithm proposed in (POPA *et al.*, 2015). More precisely, given a threshold γ, let $Edges_{ST_{j,i}} = \{(e_1, e_2, ..., e_n)\}$ be the set of the edges in $ST_{j,i}$. We claim that $ST_{j,i}$ can be represented by means of a network path $np \in NP$ if and only if

$$\frac{\left|Edges_{ST_{j,i}} \cap np\right|}{|np|} \geq \gamma, \tag{5.2}$$

and $Edges_{ST_{j,i}} \cap np$ must be a subpath of np and a subpath of $Edges_{ST_{j,i}}$.

When applying the compression over the original sub-trajectories we preserve the information concerning the time domain. For instance, let us suppose that we have $ST_{j,i} = \{(e_1,(t_1,t_2)),(e_2,(t_2,t_3)),(e_3,(t_3,t_4)),(e_4,(t_4,t_5))\}$ and two network paths, $np_1, np_2 \in NP$, where $np_1 = \{(e_1,e_2)\}$ and $np_2 = \{(e_3,e_4)\}$; then, we can say that $ST_{j,i} = \{(np_1,(t_1,t_3)),(np_2,(t_3,t_5))\}$, i.e., during the time interval (t_1,t_3) o_j crossed np_1, while during (t_3,t_5) o_j crossed np_2.

At this point we can introduce a distance function that is able to compare sub-trajectories while leveraging the concept of network paths. The rationale is to *save* a relevant amount of computation time, since it avoids the usage of the shortest path algorithm during the clustering.

First, we need to introduce the *Network Path Similarity* function. The idea behind this function is to take in input two sub-trajectories and determine if the two sub-trajectories cross the same network path within close temporal intervals.

Definition 5.1.2. (**Network Path Similarity**) Let $ST_{j,i}$ and $ST_{k,i}$ be two sub-trajectories associated with the objects o_j and o_k at time window i. Let also δ be a time tolerance and t a timestamp. Then, we define the Network Path Similarity (NPS) function to be a boolean funcion $NPS : ST \times ST \times R_{\geq 0} \times R_{\geq 0} \rightarrow \{0,1\}$ such that:

$$NPS(ST_{j,i}, ST_{k,i}, \delta, t) = \begin{cases} 1 & \exists p,\, ST_{j,i}(t) = ST_{k,i}(p) \ \wedge p \in [t-\delta, t+\delta] \\ 0 & \text{otherwise} \end{cases}$$

Thanks to Definition 5.1.2, we are now able to introduce the definition of the distance function that can be used within the micro-group maintenance phase and the clustering algorithm.

Definition 5.1.3. (**Spatio-Temporal Distance**) Given a time window $i = [t, t+\delta t]$ and let $n = min(ST_{j,i}.length, ST_{k,i}.length)$, the *spatio-temporal distance* between two sub-trajectories $ST_{j,i}$ and $ST_{k,i}$ within a road network is defined as follows:

$$distance(ST_{j,i}, ST_{k,i}, \delta) = \frac{\sum_{r=1}^{n} NPS(ST_{j,i}, ST_{k,i}, \delta, t_r)}{max(ST_{j,i}.length, ST_{k,i}.length)} \tag{5.3}$$

5.1.1 Incremental Clustering Algorithm

In this section we briefly present the differences between CUTiS and NET-CUTiS when dealing with trajectory clustering. The major differences can be summarized as follows:

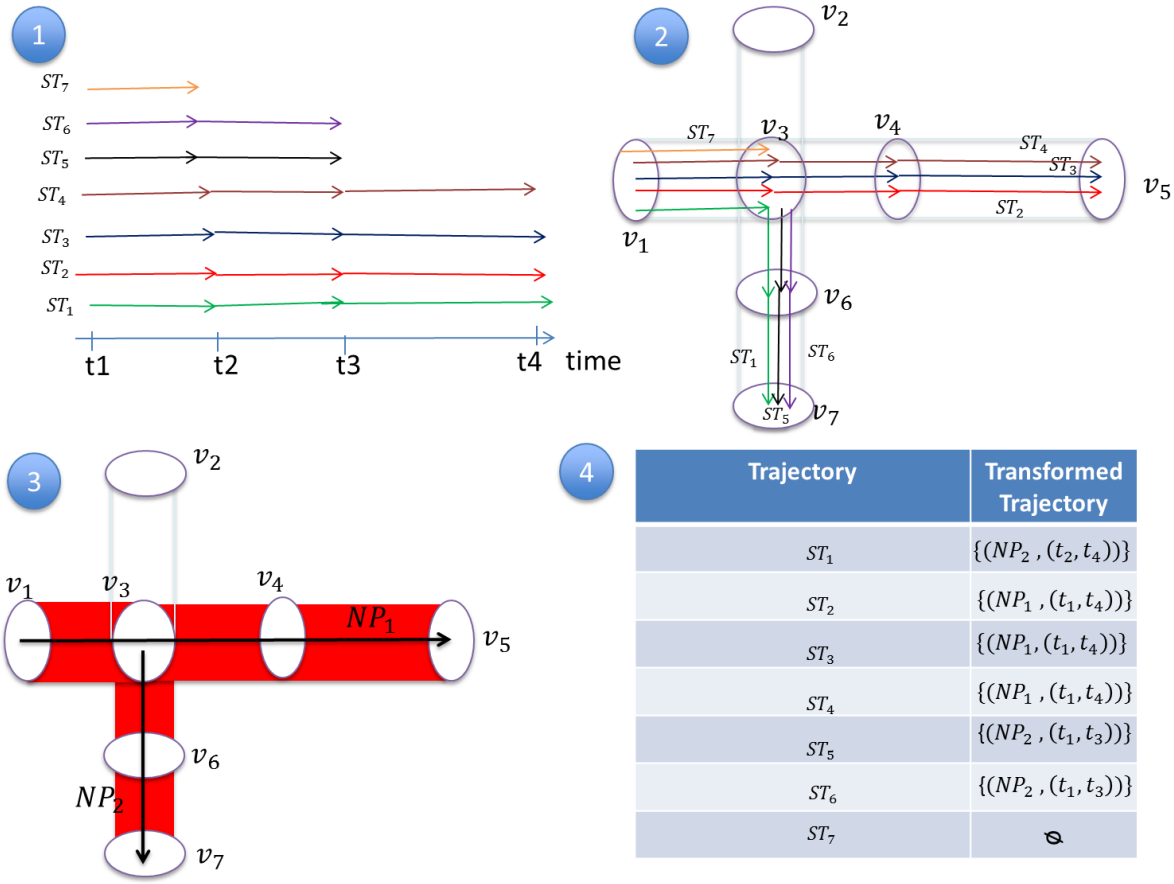
- Sub-trajectories that cannot be represented by any network path are considered as *outliers* by NET-CUTiS;
- The first step of the clustering algorithm presented in Section 4.3 is to discover merge candidates before merging pairs of micro-groups. This operation inevitably has to be modified when the underlying space is a road network, since the concept of micro-group *radius* loses its meaning. For the moment, we do not compute the merge candidates in the clustering phase for Net-CUTiS.

5.2 Net-CUTiS Example

Let us consider a road network $G(V,E)$, where $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ and $E = \{(v_1, v_3), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_3, v_6), (v_6, v_7)\}$. Figure 23 shows the sub-trajectories $\{ST_1, ST_2, ST_3, ST_4, ST_5, ST_6, ST_7\}$ map matched on G. Each sub-trajectory contains positions that occur within the temporal interval $[t_1, t_4]$.

The first operation carried on by Net-CUTiS is to find out the network paths within G: by means of the algorithm in (POPA *et al.*, 2015), we are able to find two network paths: $NP_1 = \{(v_1, v_3), (v_3, v_4), (v_4, v_5)\}$ and $NP_2 = \{(v_3, v_6), (v_6, v_7)\}$. As a side-note, we observe that notation previously introduced for time-windows were omitted. Indeed, detecting network paths for any time-window requires negligible computational efforts.

The next step is to represent the original sub-trajectories by means of the network paths; let us remember that, according to Definition 5.1.1, part of a sub-trajectory can be represented by means of a network path if (i) the sub-trajectory possesses at least 70% of the



Trajectory	Transformed Trajectory
ST_1	$\{(NP_2, (t_2, t_4))\}$
ST_2	$\{(NP_1, (t_1, t_4))\}$
ST_3	$\{(NP_1, (t_1, t_4))\}$
ST_4	$\{(NP_1, (t_1, t_4))\}$
ST_5	$\{(NP_2, (t_1, t_3))\}$
ST_6	$\{(NP_2, (t_1, t_3))\}$
ST_7	∅

Figura 23 – Network Paths Discovery Process

edges making up the network path, and (ii) such edges appear in the same order as in the network path: for instance, $ST_1 = \{((v_1, v_3), (t_1, t_2)), ((v_3, v_4), (t_2, t_3)), ((v_4, v_5), (t_3, t_4))\}$ possesses only 25% *of* NP_1*'s edges*, while it possesses 100% *of* NP_2*'s edges*. Accordingly, Net-CUTiS represents ST_1 as $\{(NP_2, (t_2, t_4))\}$, i.e., ST_1 crosses NP_2 during the time interval $[t_2, t_4]$. On the other hand, ST_7 represents an outlier, since it cannot be represented by any network path.

Let us suppose that we use the following parameters for micro-groups initialization and maintenance: (i) $\varepsilon = 0.5$, (ii) τ=2, and (iii) δ equals to one time unit. Overall, from Figure 24 we see that Net-CUTiS finds two micro-groups, $g_1 = \{ST_2, ST_3, ST_4\}$ and $g_2 = \{ST_1, ST_5, ST_6\}$. For what is related to the first micro-group of sub-trajectories, g_1, ST_3 is chosen to be its representative: as such, our approach goes on to compute the spatio-temporal distance between ST_3 and the other sub-trajectories, according to the function introduced in Definition 5.1.3; as a result, micro-group g_1 is found to be equal to $\{ST_2, ST_3, ST_4\}$. Next, a second micro-group, g_2, is created by using ST_6 as its representative. By examining the similarity between ST_6 and the remaining sub-trajectories, we have that it is equal to 1 when considering ST_5, while it is equal to ≈0,67 when considering ST_1; in the latter case, we note that ST_1 and ST_6 cross the same network

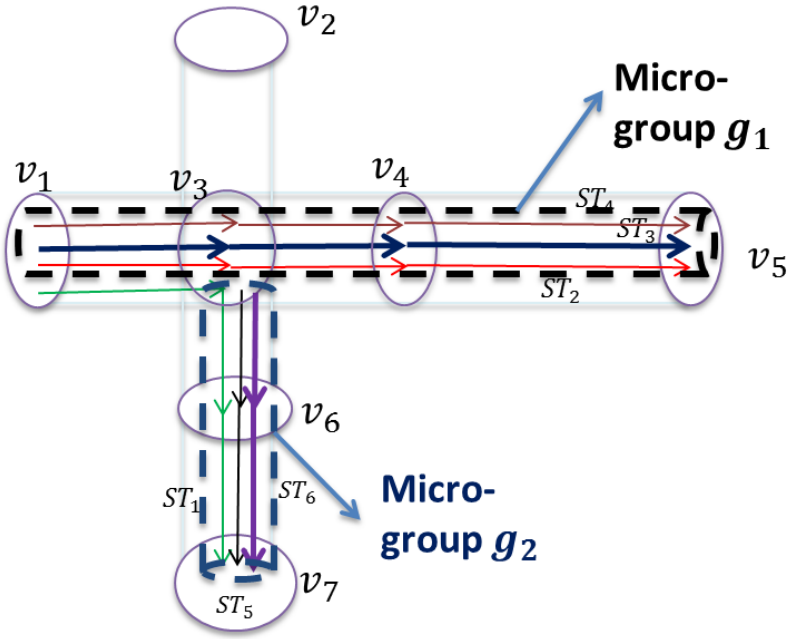
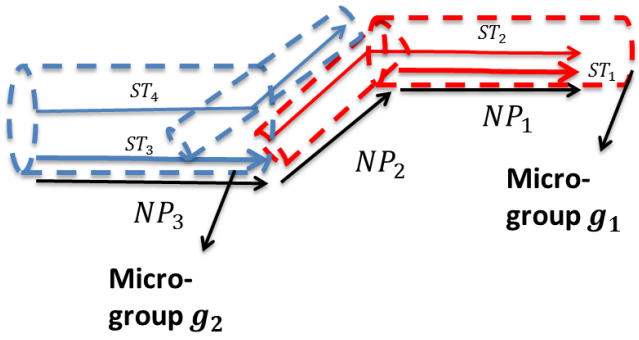


Figura 24 – Micro-groups Discovery on Road Network



Sub-Trajectory	Sub-Trajectory Represented by Dense Paths
ST_1	$\{(NP_1, (t_1, t_4))\}$
ST_2	$\{(NP_2, (t_1, t_2)), (NP_1, (t_2, t_4))\}$
ST_3	$\{(NP_2, (t_1, t_2))\}$
ST_4	$\{(NP_3, (t_1, t_2)), (NP_2, (t_2, t_3))\}$

Figura 25 – Clustering Sub-trajectory on Road Network by merging Micro-groups

path, NP_2, yet at different time-stamps: however, once we take into account the time tolerance, δ, both sub-trajectories are deemed similar in the temporal domain as well. Consequently, g_2 is found to be equal to $\{ST_1, ST_5, ST_6\}$.

All in all, we observe that each representative sub-trajectory gives a *succinct* representation of the movements characterizing the set of sub-trajectories included in its micro-group: this is achieved by taking into account both the spatial domain – we look at the network paths shared by the sub-trajectories – and the temporal domain – we make sure that sub-trajectories cross the aforementioned network paths within close temporal intervals.

We conclude the example by showing how Net-CUTiS *merges* different micro-groups to create clusters of sub-trajectories [1]. More specifically, we remind that Net-CUTiS checks the density connection between representative trajectories to merge two or more micro-groups.

[1] We remind that a single micro-group may represent a cluster as well.

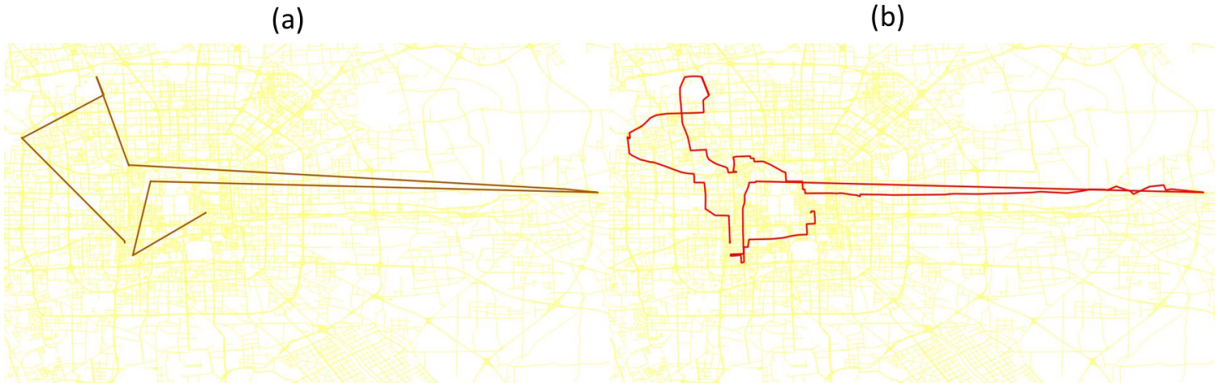


Figura 26 – Difference between a sub-trajectory (a) before adding the shortest path between each two consecutive positions and (b) after

Going back to the example, let us consider three network paths, NP_1, NP_2 and NP_3 (Figure 25, highlighted in black), and the representative trajectories of micro-groups g_1 (red) and g_2 (blue), i.e., ST_1 and ST_3 respectively. According to the parameters specified previously, we have that $N_\varepsilon(ST_1) = \{ST_1, ST_2\}$, $N_\varepsilon(ST_2) = \{ST_1, ST_2, ST_4\}$, $N_\varepsilon(ST_3) = \{ST_3, ST_4\}$ and $N_\varepsilon(ST_4) = \{ST_2, ST_3, ST_4\}$. As such, micro-groups g_1 and g_2 merge to create a cluster, since there is a density connection between their representative trajectories: $\{ST_1, ST_2, ST_4, ST_3\}$.

The next section discusses our experiments for Net-CUTiS, NETSCAN and DBSCAN.

5.3 Experiments

In order to assess the Net-CUTiS approach, the same dataset scenarios discussed in Section 4.6.3 have been used. First, we need to report a pre-processing phase in the data set since for a sub-trajectory, two consecutive sample positions are not in continuous edges of the road network. Then, for each sub-trajectory, between two consecutive sample positions, the sub-trajectory was fed with the shortest path on the road network between these positions. Figure 26 presents the difference between including the shortest path between two consecutive positions and keep the data as it is.

Throughout these experiments, we set the Net-CUTiS parameters for the clustering algorithm equals to ε=0.5, τ=10, ρ=0.5 and δ=5 minutes. Moreover, there exists another parameter γ used in the compression processing of sub-trajectories into network paths. However, the choice of γ value has a trade-off between forcing the sub-trajectories to be almost totally represented by the dense paths and the need to represent many sub-trajectories in the data set (this means less outliers in the clustering). For both scenarios, we have chosen to set γ=0.7

The paper (KHARRAT *et al.*, 2008) proposes NETSCAN which is our competitor approach. NETSCAN finds density based clusters for network constrained trajectories. At first, NETSCAN finds the dense paths, i.e. a sequence of neighbor road segments. The minimal required density for a road segment is α moving objects crossing it. The maximal density difference between neighbor segments that belong to the same dense path is equal to ε. NETSCAN does not ensure that a dense path is crossed by the same moving object trajectories, unlike the Definition 5.1.1 of network paths. NETSCAN trajectory clustering algorithm consists in grouping the trajectories according to their similarity to each dense path generated. For each dense path, it computes the similarity with each trajectory. If the similarity is above the threshold value σ, then the trajectory is kept in the cluster. More precisely, the common part between the trajectory and the dense path is added to the cluster. In this way, the number of returning clusters is equal to the number of dense paths.

There are three main drawbacks of using NETSCAN approach compared using Net-CUTiS: (1) The network paths are semantically well-related to the clustering idea: grouping the sub-trajectories which share within close temporal interval; (2) Furthermore, the same moving object trajectory can be associated to different clusters in NETSCAN. This is different compared to DBSCAN clusters which our approach is based on, a moving object sub-trajectory can only be associated to one single cluster; (3) NETSCAN does not detect the clusters evolution online.

We believe the clusters found by using Net-CUTiS are more similar to traveling together patterns such as families staying together or by considering each person as a moving object and a group of person sharing the same trip using a public transport. We avoid comparing the effectiveness between Net-CUTiS and NETSCAN, since the semantics of their clusters results are not exactly the same. In these experiments, we executed NETSCAN from scratch for each time window and we set its parameters equal to σ=0.5, α=10 and ε=1.

There are two ways to perform the experiments with Net-CUTiS. First, the network paths can be discovered from a historical dataset (batch mode) or online at each time window. When we experimented with a historical dataset from T-DRIVE which holds around 4,000 moving object trajectories during 1 hour, the running time was 183,719 seconds. Figure 27 shows the found network paths. For online discovery of network paths at each time window, the running time was on average 132,1804 seconds. By executing the network path discovery in batch mode, the quality of the network paths is higher, since it creates less network paths described by a long sequence of roads. Moreover, it is less time-consuming.

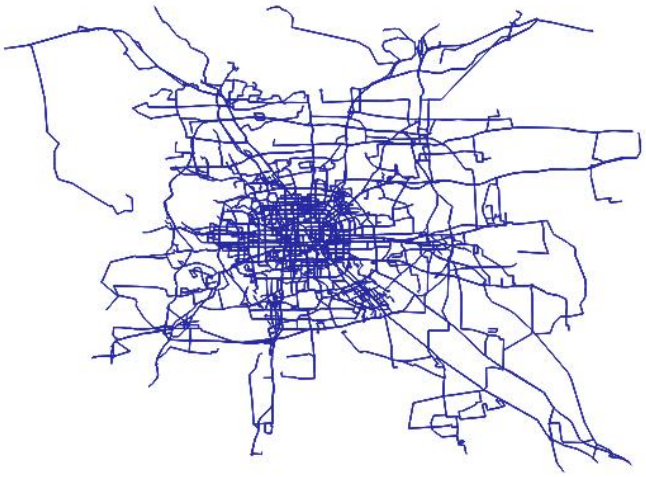


Figura 27 – Network paths from 17:30 to 18:30 on 4th of February 2008 for "densified"T-DRIVE dataset

Figure 28 reports the distribution of the cardinality of the network paths found. The cardinality of a network path corresponds to its number of edges or road segments. We also found few network paths with more than 37 different road segments and less than 339. However, we omit in Figure 28 in order to ease viewing. Figures 29 and 30 report the dense paths cardinality distribution found by NETSCAN during the 5 consecutive time windows.

Efficiency Analysis. Figures 31 and 32 present the running time of Net-CUTiS approach for the both scenarios comparing to NETSCAN (KHARRAT *et al.*, 2008). The running time is computed by piling the cost of the following steps: (1) the network path discovery (only computed in the first time window since we have used a historical trajectory dataset); (2) the compression of moving object sub-trajectories into network paths; (3) the micro-group maintenance and clustering. The performance of Net-CUTiS has improved compared to CUTiS* (see Figure 21). The main reason is the moving object sub-trajectories are compressed and represented by the network paths. Then, the pairwise sub-trajectory comparison does not need to scan all sub-trajectory positions. By using the network path representation, Net-CUTiS compares the sub-trajectories in a higher granularity which speeds up the processing time. Note that Net-CUTiS outperforms our competitor NETSCAN as well (KHARRAT *et al.*, 2008).

NETSCAN discovered in average 3,544 dense paths for the Scenario I and 3,677 dense paths for the Scenario II. Net-CUTiS discovered only 833 network paths for both scenarios

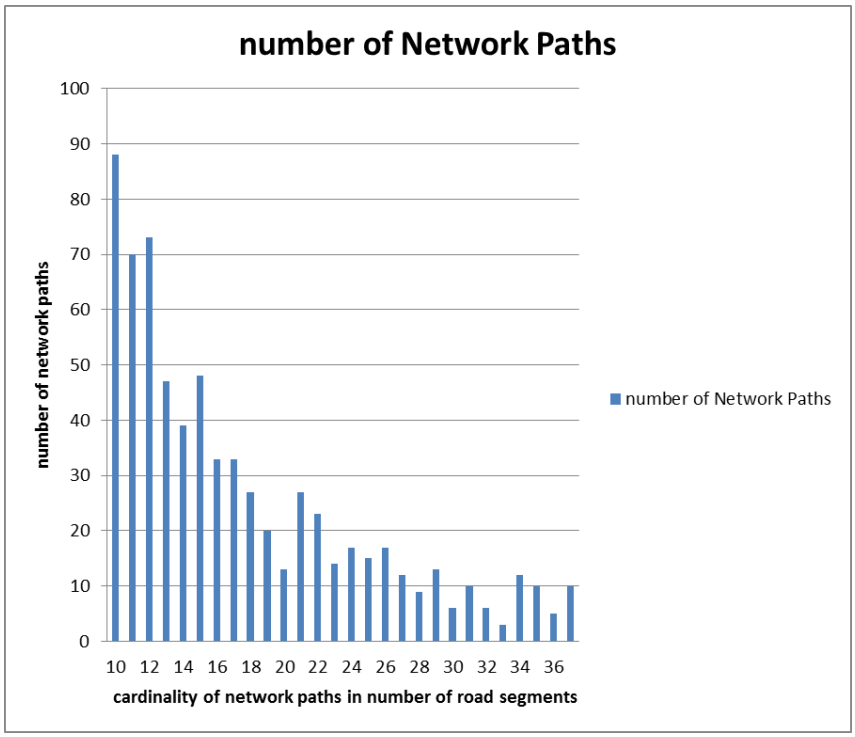


Figura 28 – Network Path Cardinality Distribution

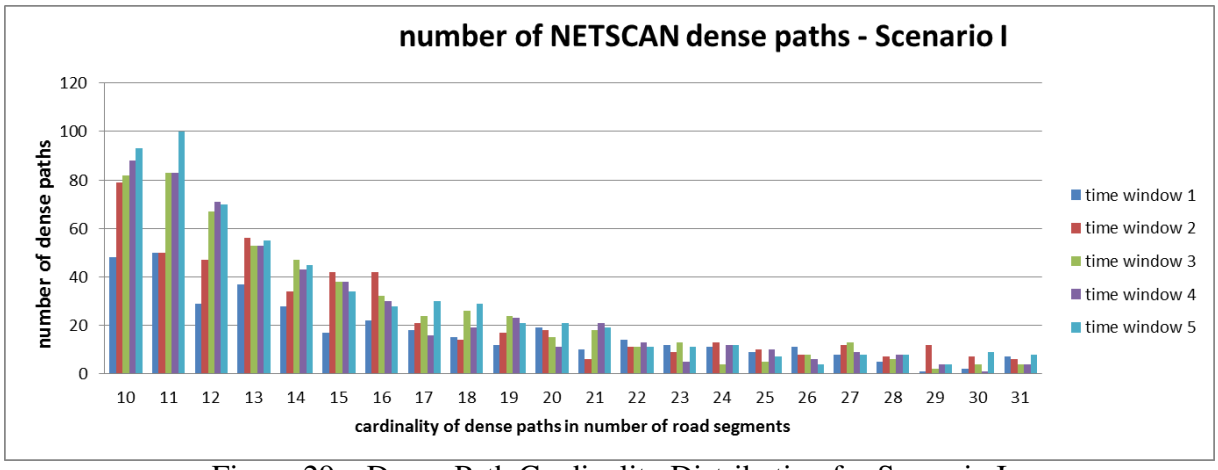


Figura 29 – Dense Path Cardinality Distribution for Scenario I

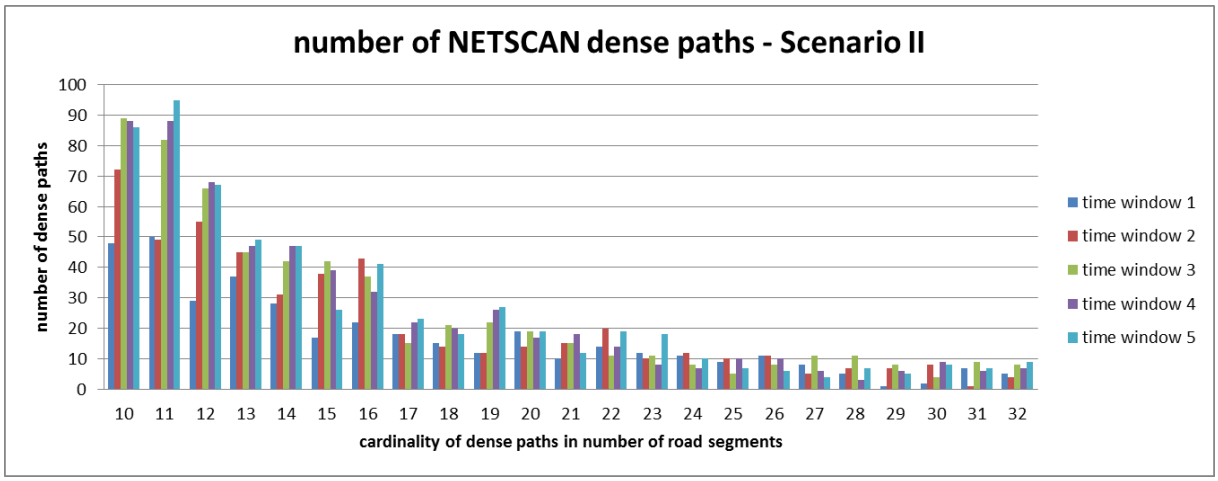


Figura 30 – Dense Path Cardinality Distribution for Scenario II

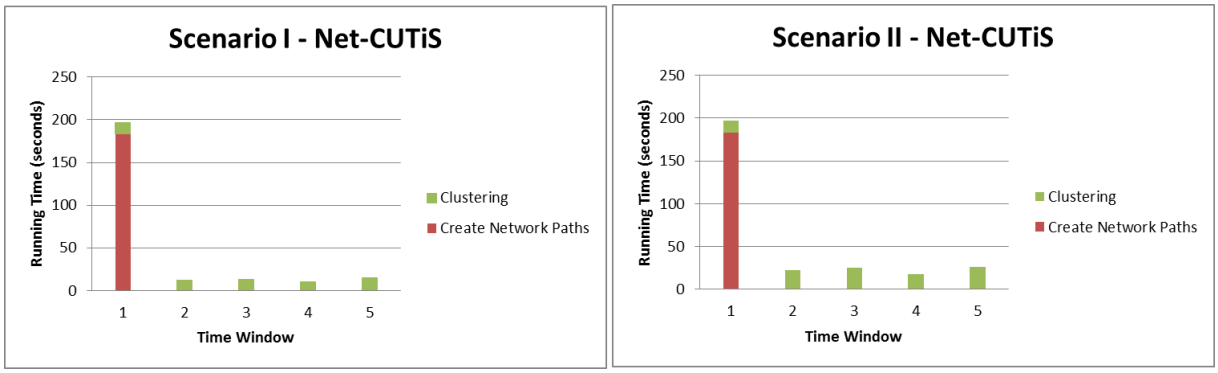


Figura 31 – Efficiency Analysis: Sub-trajectory Clustering Discovery using Net-CUTiS.

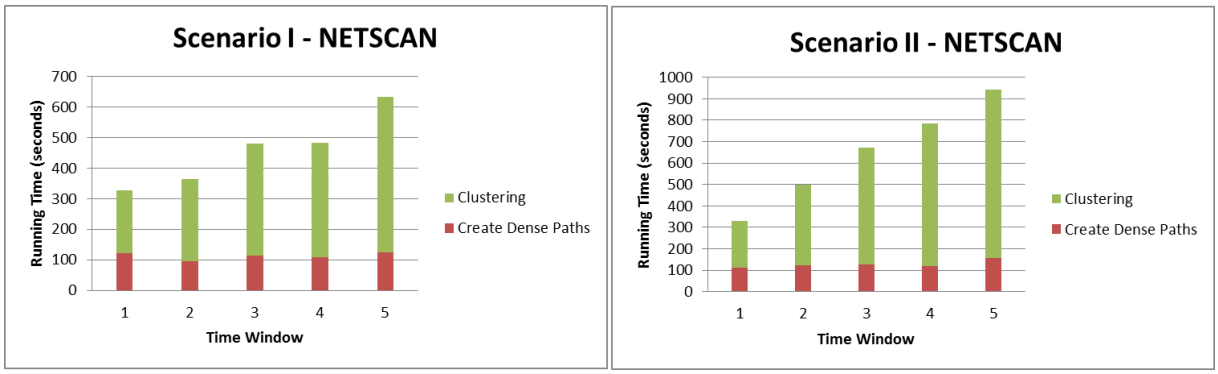


Figura 32 – Efficiency Analysis: Sub-trajectory Clustering Discovery using NETSCAN.

evaluated, which is less than the number of dense paths found by NETSCAN. Furthermore, our clusters might be composed of moving object sub-trajectories represented by one or more network paths. On the other hand, NETSCAN has only one dense path per cluster. That's why NETSCAN discovered more clusters than Net-CUTiS in these experiments. The number of clusters found by NETSCAN and Net-CUTiS are reported in the Figure 33.

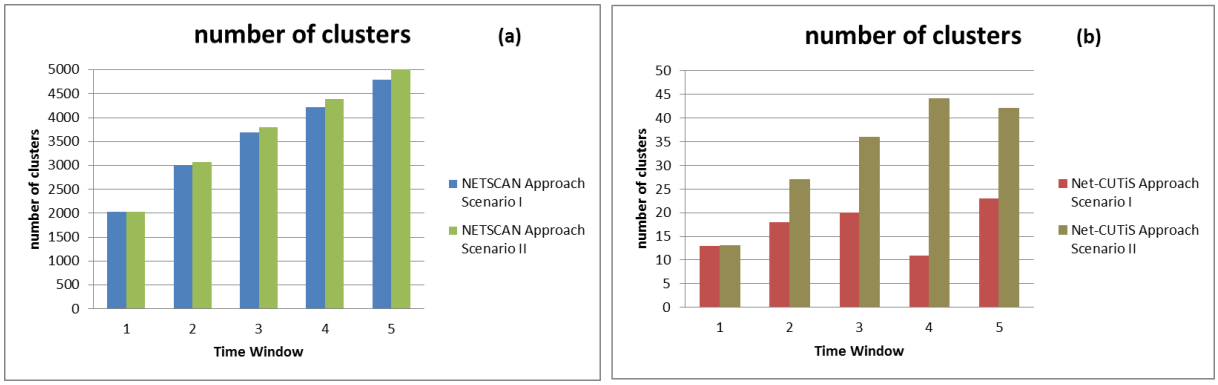


Figura 33 – (a) Number of Sub-trajectory Clusters found by NETSCAN and (b) Net-CUTiS for the Scenario I and Scenario II.

We also compared the efficiency of Net-CUTiS with DBSCAN. In DBSCAN implementation, we applied the Synchronous Shortest Path Distance as a distance function. We refer to the Definition 4.2.1 in order to explain the Synchronous Shortest Path Distance (in fact, we

just replaced the Euclidean Distance by the Shortest Path Distance). As we mentioned before, the shortest path distance computation is computationally costly. Figure 34 reports the comparison between DBSCAN and Net-CUTiS by using the dataset of the Scenario II. Net-CUTiS takes far less time than DBSCAN in the 5 consecutive time windows.

We implemented an index structure to accelerate range queries from a moving object sub-trajectory in the DBSCAN execution. Basically, for each sub-trajectory position we stored in a B-tree the 3-tuple (road segment id, timestamp, sub-trajectory id). In this way, for each sub-trajectory, we can query the B-tree by searching all sub-trajectories id where the Euclidean distance between their road segments would be less or equal to ε at the same timestamp. In this way, a moving object sub-trajectory and its possible neighbors sub-trajectories have positions close in space and time. Note that in the road network space, if the Euclidean distance between two objects is larger than the threshold ε, these objects are not neighbors by using the Shortest Path distance as a distance measure. After retrieving the neighbors candidates from the index, our implementation computes the Synchronous Shortest Path distance in order to verify the real neighbors.

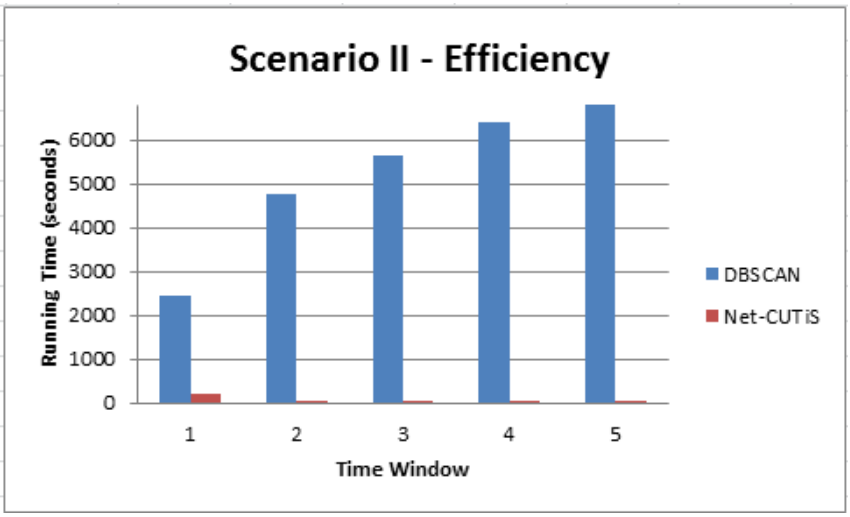


Figura 34 – For the Scenario II, the execution time of running DBSCAN with Synchronous Shortest Path distance versus Net-CUTiS.

Effectiveness Analysis. Since Net-CUTiS produces clusters semantically different (according to network path representation (POPA *et al.*, 2015)) from others clustering approaches, they have no common quantitative measure of the accuracy. Therefore, we evaluated the accuracy of our approach by visual inspection as already did in the papers (LEE *et al.*, 2007; LI *et al.*, 2007; HAN *et al.*, 2012; ESTER *et al.*, 1996), among others.

Figure 38 presents the sub-trajectory clusters found in three consecutive time win-

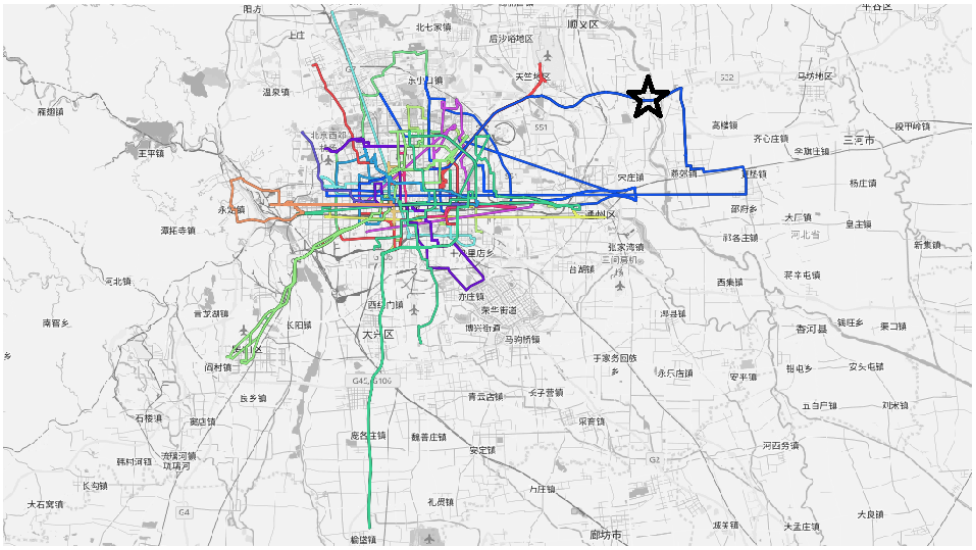


Figura 35 – Representative Trajectories for the 1st Time Window

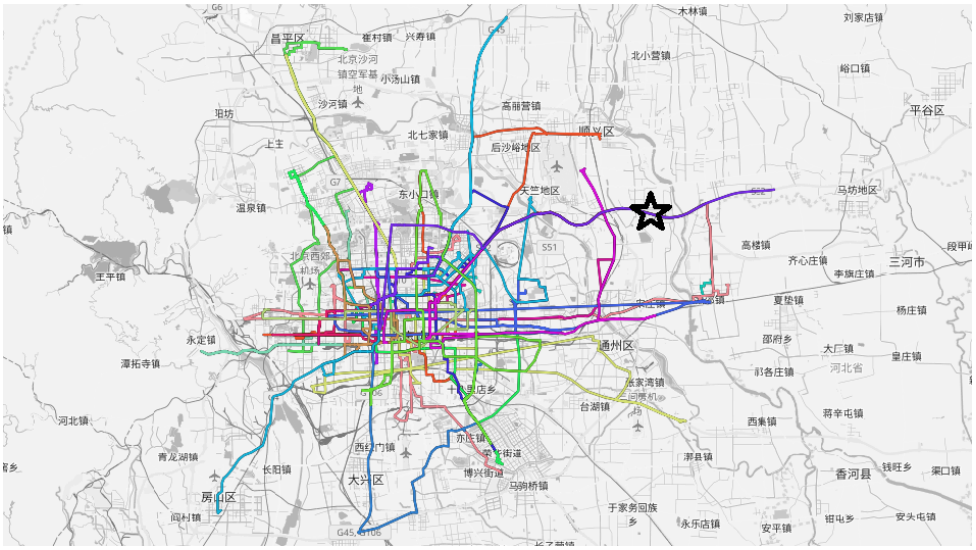


Figura 36 – Representative Trajectories for the 2nd Time Window

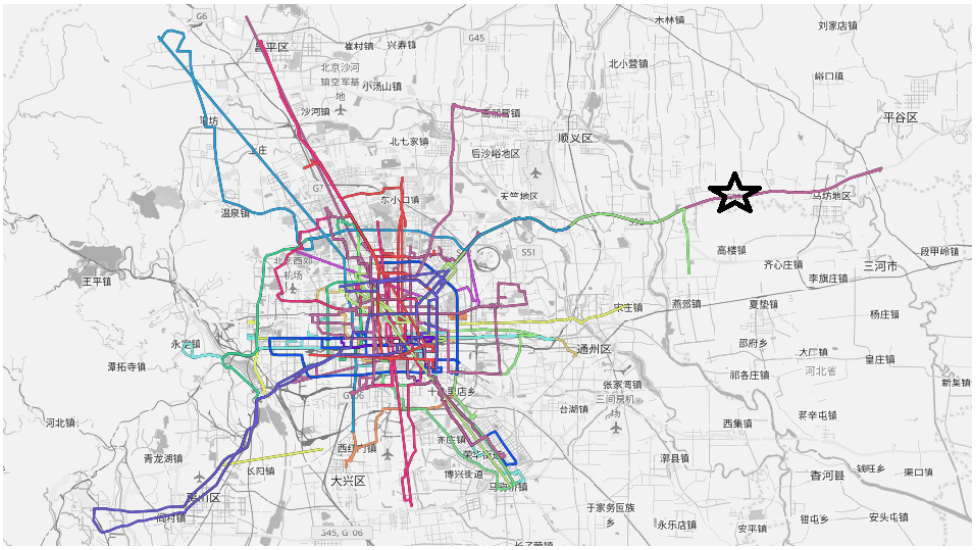


Figura 37 – Representative Trajectories for the 3rd Time Window

Figura 38 – Representative Trajectories of each cluster found by Net-CUTiS using Scenario I dataset for three consecutives time windows.

dows using the same dataset of Scenario I. Even compared to TraClus and CUTiS*, Net-CUTiS produces clusters with longer representative routes: this may be useful to improve the services offered by location based applications, for instance those that have to deal with the optimization of bus lines, or offer ride-sharing services (KAMAR; HORVITZ, 2009). In comparison with Figure 19 in Section 4.6.3, Net-CUTiS produces more compact and meaningful results through a road network aware sub-trajectory clustering.

The representative trajectories of the clusters in the Figure 38 represent the road segments where traffic is highly concentrated. You can easily see the sub-trajectories that belong to the found clusters were crossing roads very close to airports. Moreover, there is one cluster for all time windows that crossed S32 Jingping Expressway (an expressway in Beijing. We indicated this expressway with a star in the Figure). Observe how the clusters on S32 Jingping Expressway continued to lengthen from the first time window to the third. It might indicate the increasing flow in the network paths of the Expressway. By utilizing the longer network paths representation to compress the sub-trajectories and to measure the similarity in the clustering algorithm, Net-CUTiS not only produces meaningful sub-trajectory clusters but also runs very fast.

We also measure the effectiveness of Net-CUTiS using another real dataset from a taxi fleet company of Fortaleza, Brazil. This dataset has around 400 different moving objects. Then, around 400 moving object trajectories at each observed time window. It is a small dataset, but it is still dense because the sampling rate is equal to 1 second. The taxi fleet belongs to a private company that tags and tracks its vehicles as they move around, and it aims at supporting drivers' decisions related to displacement through the city in (quasi) real time (i.e., movement behaviors through the city that can be found by sub-trajectory cluster analysis).

For these experiments, we used the time window size equals to 10 minutes. The taxi fleet company has the majority of its taxis offering the service in popular districts of Fortaleza. This is what they call Popular Taxi Service. Figure 41 shows the representative trajectories of the clusters found in three residential districts of Fortaleza: Mondumbi, Conjunto Esperança and Prefeito José Walter. As already expected, these regions of Fortaleza covered by Net-CUTiS sub-trajectory clusters are very popular neighborhoods. Comparing both time windows, we can easily see the increased movement of moving objects sub-trajectories in the Perimetral Avenue and Bernardo Manuel Avenue. These are the two main routes in Mondumbi district.

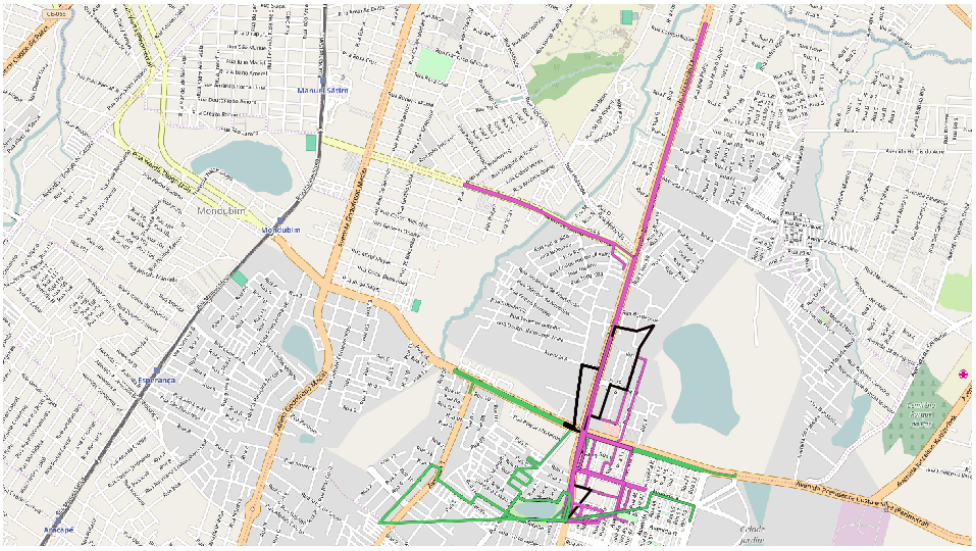


Figura 39 – Representative Trajectories for the 1st Time Window

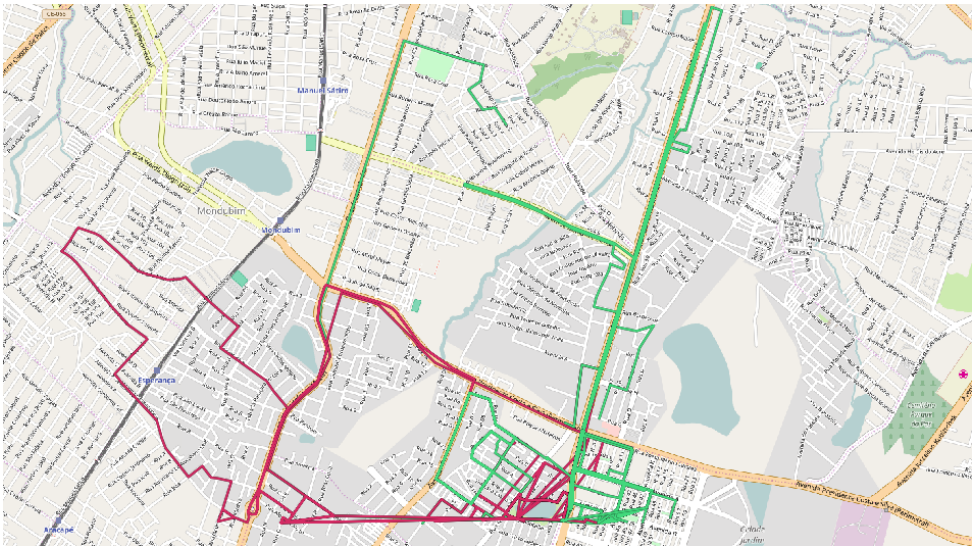


Figura 40 – Representative Trajectories for the 2nd Time Window

Figura 41 – Representative Trajectories of each cluster found by Net-CUTiS using Taxi dataset of Fortaleza, Brazil.

5.4 Discussion

In this chapter, we comprehensively discussed Net-CUTiS, the extension of our framework CUTiS for constrained movement in the road network space. Net-CUTiS addresses the same problem presented in the last section, how to online track moving objects and find the sub-trajectory clusters without having to cluster from scratch. However, in this Chapter we considered the movement on the road network. The main contribution of this Chapter is the proposal of a new trajectory distance function which is based on first describing the trajectories into a set of network paths (POPA *et al.*, 2015) and then compare if two sub-trajectories cross the same network paths within a close time interval.

Our experiments were conducted on real data sets from Taxis of Beijing, and it shows the efficiency of Net-CUTiS compared to NETSCAN (KHARRAT *et al.*, 2008) and DBSCAN. We also have an improvement in the efficiency by comparing to CUTiS which uses Synchronous Euclidian Distance to measure the similarity between two sub-trajectories. Net-CUTiS saves computation time, since comparing the distance between two sub-trajectories means comparing the network path(s) identification(s). We also compared the effectiveness of our approach by using visual inspection for two real datasets.

6 CONCLUSION

6.1 Summary of thesis contributions

Moving objects describe the continuous evolution of spatial objects over time. One feature of moving objects is that their locations and velocity change continuously with time, and these are reported in the format of trajectory streams, making them more complicated than static spatial objects (LIU; SCHNEIDER, 2011). Nowadays, many services track and report moving object trajectories (e.g., of persons, vehicles, or animals) continuously (e.g., every second or every minute). By analyzing these data in a real time fashion, we bring a real added-value in the comprehension of city dynamics, and the detection of regularities as well as anomalies, which is essential for decision-making. Among these patterns, we mainly considered in this thesis the (sub)-trajectory clustering and its evolution. Another problem addressed in this thesis was how to efficiently maintain these patterns as the moving object trajectories evolve in time. Furthermore, we studied our problem in both: Euclidean space and on the Road Network.

First, we proposed a model to describe the cluster evolution pattern and an algorithm to detect these patterns. We also proposed an algorithm to discover frequent evolution patterns. CUTiS is the major contribution of this thesis and an improvement of the first proposal discussed in Chapter 3, since, in the first approach, we needed to cluster the data from scratch. CUTiS addresses the problem of discovering and maintaining sub-trajectory clusters and its evolution at each time window, since, as each moving object updates its sub-trajectory from time to time, new moving objects appear and others disappear from the system. As each moving object evolves in time, also the cluster evolves. We do not aim at computing the clusters from scratch since it is computationally expensive. There is a need to online track trajectory clusters over a sliding window. CUTiS incrementally maintains each group of objects which moves together (called micro-groups) and captures its evolution patterns from time to time. By merging micro-groups, CUTiS discovers density based sub-trajectory clusters. Our experiments were conducted on real data sets from Taxis in Beijing and Taxis in Fortaleza city, and they show the efficiency and effectiveness of CUTiS compared to our competitors.

In order to improve the efficiency of our proposal, we included a sub-trajectory index in CUTiS. In this way, we proposed CUTiS*. The proposed index improves the performance without losing quality in the cluster results, as we show in our experiments conducted on a real dataset of Taxis of Beijing. Another contribution of this thesis is a proposed demo tool which

implements CUTiS* for online mobility pattern discovery in sub-trajectory data streams. The demo is implemented for patterns such as micro-groups, density-based sub-trajectory clusters, and trajectory flocks. The other mobility patterns such as convoys and swarms, among others, will be implemented in the future.

We also proposed Net-CUTiS, the extension of our framework CUTiS for road network movement constraint. Net-CUTiS tracks moving objects and finds the sub-trajectory clusters without having to do cluster from scratch. However, it considers the trajectories' movement constrained to the road network. In this way, we proposed a new trajectory distance function which is based on first describing the trajectories into a set of network paths (POPA *et al.*, 2015) and then comparing them to see if two trajectories cross the same network paths with similarity in time as well.

Another contribution investigated in this thesis was a proposed a version of the DBSCAN algorithm in a distributed way by using the MapReduce paradigm. This is a first step of the purposes of distributing CUTiS* and Net-CUTiS in order to achieve a more efficient trajectory clustering algorithm.

Below, we list all the publications derived from the work which was done during this thesis.

Conferences

1. Ticiana Linhares Coelho da Silva, Antônio Cavalcante de Araújo Neto, Régis Pires Magalhães, Victor A. E. de Farias, José Antônio Fernandes de Macêdo, Javam de Castro Machado. Efficient and Distributed DBSCAN Algorithm Using MapReduce to Detect Density Areas on Traffic Data. In: 16th International Conference on Enterprise Information Systems, 2014, Lisbon. p. 52.
2. Ticiana Linhares Coelho da Silva, Karine B. Zeitouni, José Antônio Fernandes de Macêdo. Online Clustering Trajectory Data Stream. In: International Conference on Mobile Data Management, 2016.
3. Ticiana Linhares Coelho da Silva, Karine B. Zeitouni, José Antônio Fernandes de Macêdo, Marco Antônio Casanova. On-Line Mobility Pattern Discovering using Trajectory Data. In: 19th International Conference on Extending Database Technology, 2016.
4. Ticiana Linhares Coelho da Silva, Karine B. Zeitouni, José Antônio Fernandes de Macêdo, Marco Antônio Casanova. CUTiS*: optimized online ClUstering of Trajectory data Stream. In: International Database Engineering & Applications Symposium, 2016.

Demo

1. Ticiana Linhares Coelho da Silva, Karine B. Zeitouni, José Antônio Fernandes de Macêdo, Marco Antônio Casanova. A Framework for Online Mobility Pattern Discovery from Trajectory Data Streams. In: International Conference on Mobile Data Management, 2016.

Workshops

1. Antônio Cavalcante de Araújo Neto, Ticiana Linhares Coelho da Silva, Victor A. E. de Farias, José Antônio Fernandes de Macêdo, Javam de Castro Machado. G2P: A Partitioning Approach for Processing DBSCAN with MapReduce. In: Web and Wireless Geographical Information Systems. Springer International Publishing, 2015. p. 191-202.
2. Ticiana Linhares Coelho da Silva, José Antônio Fernandes de Macêdo, Marco Antônio Casanova. Discovering frequent mobility patterns on moving object data. In: the Third ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems - MobiGIS '14. New York: ACM Press, 2014. p. 60.

Book chapters

1. Ticiana Linhares Coelho da Silva, Antônio Cavalcante de Araújo Neto, Régis Pires Magalhães, Victor A. E. de Farias, José Antônio Fernandes de Macêdo, Javam de Castro Machado. Towards an Efficient and Distributed DBSCAN Algorithm Using MapReduce. In: Enterprise Information Systems (pp. 75-90). Springer International Publishing, 2015.
2. Ticiana Linhares Coelho da Silva, Flávio R. C. Sousa, Antônio Cavalcante de Araújo Neto, José Antônio Fernandes de Macêdo, Javam de Castro Machado. Análise em Big Data e um Estudo de Caso utilizando Ambientes de Computação em Nuvem. In: Simpósio Brasileiro de Banco de Dados. (Org.). 1ed.Porto Alegre: SBC, 2013, v.1, p. 1-30.

6.2 Open Directions

As open directions, we will continue to investigate research problems related to this thesis. We present a few of them below:

1. Since we used the algorithm of (POPA *et al.*, 2015) to discover network paths (Definition 5.1.1) and compress our trajectories with them, we aim at proposing in the future an incremental algorithm to discover online the network paths. The algorithm proposed by (POPA *et al.*, 2015) is for historical datasets and does not observe how these network paths might change as time goes by. This perspective work is an open issue related to the **Challenge 4** of this thesis;

2. By proposing an incremental algorithm to maintain the network paths on the road network, we aim at proposing an online compression technique for trajectory data streams. This perspective work is an open issue related to the **Challenge 4** of this thesis;
3. We aim at studying how we can model and discover other mobility patterns such as convoys and swarms, gathering by maintaining micro-groups as we did for trajectory flocks and density-based trajectory clusters. This perspective work is an open issue related to the **Challenge 1** of this thesis;
4. In order to facilitate the micro-group maintenance, we aim at investigating machine learning algorithms to predict the next positions of a moving object since all the tracked objects change their position as time goes by. Furthermore, we aim to predict moving objects' evolution as well as cluster evolution. This perspective work is an open issue related to the **Challenge 2** of this thesis;
5. We aim to experiment with different distance functions for trajectories by comparing their efficiency and effectiveness in trajectory clustering algorithms. This perspective work is an open issue related to the **Challenges 1, 2, 3 and 4** of this thesis;
6. We still aim to improve the efficiency of CUTiS* in order to better support potential applications in the real world-business scenarios and captured patterns and their evolutions in time for objects that move in Euclidean space. We aim at supporting potential applications which need to track moving objects, such as a company which needs to track how its taxi fleet is moving around the city. This perspective work is an open issue related to the **Challenge 2** of this thesis;
7. We aim to provide integration of Net-CUTiS to the Graphast framework. Graphast is a framework tool that allows developers to compose a number of network models, data importing/exporting services, as well as query services, in order to quickly build applications on time-dependent networks. The main goal is to allow developers to implement solutions to different types of problems on time-dependent networks using spatial queries, such as nearest neighbor queries and optimal sequenced routes (MAGALHÃES *et al.*, 2015). Graphast does not present any mobility pattern discovery implementations for trajectory data. This perspective work is an open issue related to the **Challenge 4** of this thesis;
8. We aim to distribute and paralellize CUTiS* and Net-CUTiS in order to propose a solution for large scale mobility data stream.
9. We aim to enhance the CUTiS algorithm to deliver good results without requiring parame-

ters' configuration from the users.

REFERÊNCIAS

AGGARWAL, C. C.; HAN, J.; WANG, J.; YU, P. S. A framework for clustering evolving data streams. In: VLDB ENDOWMENT. **Proceedings of the 29th international conference on Very large data bases-Volume 29**. [S.l.], 2003. p. 81–92.

AGRAWAL, R.; SRIKANT, R. *et al.* Fast algorithms for mining association rules. In: **Proc. 20th int. conf. very large data bases, VLDB**. [S.l.: s.n.], 1994. v. 1215, p. 487–499.

ANKERST, M.; BREUNIG, M. M.; KRIEGEL, H.-P.; SANDER, J. Optics: ordering points to identify the clustering structure. In: ACM. **ACM Sigmod Record**. [S.l.], 1999. v. 28, n. 2, p. 49–60.

CAO, F.; ESTER, M.; QIAN, W.; ZHOU, A. Density-based clustering over an evolving data stream with noise. In: **SDM**. [S.l.: s.n.], 2006. v. 6, p. 328–339.

CHEN, J.; LEUNG, M. K.; GAO, Y. Noisy logo recognition using line segment hausdorff distance. **Pattern recognition**, Elsevier, v. 36, n. 4, p. 943–955, 2003.

CHEN, L.; ÖZSU, M. T.; ORIA, V. Robust and fast similarity search for moving object trajectories. In: ACM. **Proceedings of the 2005 ACM SIGMOD international conference on Management of data**. [S.l.], 2005. p. 491–502.

CHEN, Y.; TU, L. Density-based clustering for real-time stream data. In: ACM. **Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2007. p. 133–142.

DAI, B.-R.; LIN, I.-C. Efficient map/reduce-based dbscan algorithm with optimized data partition. In: **Cloud Computing, 2012 IEEE 5th International Conference on**. [S.l.: s.n.], 2012. p. 59–66.

DENG, K.; XIE, K.; ZHENG, K.; ZHOU, X. Trajectory indexing and retrieval. In: **Computing with spatial trajectories**. [S.l.]: Springer, 2011. p. 35–60.

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.

ESTER, M.; KRIEGEL, H.-P.; S, J.; XU, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In: **KDD**. [S.l.: s.n.], 1996. v. 96, p. 226–231.

ESTER, M.; KRIEGEL, H.-P.; SANDER, J.; WIMMER, M.; XU, X. Incremental clustering for mining in a data warehousing environment. In: **VLDB**. [S.l.: s.n.], 1998. v. 98, p. 323–333.

FRENTZOS, E.; GRATSIAS, K.; PELEKIS, N.; THEODORIDIS, Y. Algorithms for nearest neighbor search on moving object trajectories. **Geoinformatica**, Springer, v. 11, p. 159–193, 2007.

FRENTZOS, E.; GRATSIAS, K.; THEODORIDIS, Y. Index-based most similar trajectory search. In: IEEE. **2007 IEEE 23rd International Conference on Data Engineering**. [S.l.], 2007. p. 816–825.

GIANNOTTI, F.; NANNI, M.; PINELLI, F.; PEDRESCHI, D. Trajectory pattern mining. In: ACM. **Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2007. p. 330–339.

GRÜNWALD, P. D.; MYUNG, I. J.; PITT, M. A. **Advances in minimum description length: Theory and applications**. [S.l.]: MIT press, 2005.

GUDMUNDSSON, J.; KREVELD, M. van. Computing longest duration flocks in trajectory data. In: ACM. **Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems**. [S.l.], 2006. p. 35–42.

GUDMUNDSSON, J.; KREVELD, M. van; SPECKMANN, B. Efficient detection of motion patterns in spatio-temporal data sets. In: ACM. **Proceedings of the 12th annual ACM international workshop on Geographic information systems**. [S.l.], 2004. p. 250–257.

GUHA, S.; MISHRA, N.; MOTWANI, R.; O'CALLAGHAN, L. Clustering data streams. In: IEEE. **Foundations of computer science, 2000. proceedings. 41st annual symposium on**. [S.l.], 2000. p. 359–366.

GUHA, S.; RASTOGI, R.; SHIM, K. Cure: an efficient clustering algorithm for large databases. In: ACM. **ACM SIGMOD Record**. [S.l.], 1998. v. 27, n. 2, p. 73–84.

HAN, B.; LIU, L.; OMIECINSKI, E. Neat: Road network aware trajectory clustering. In: **ICDCS**. [S.l.: s.n.], 2012. p. 142–151.

HE, Y.; TAN, H.; LUO, W.; MAO, H.; MA, D.; FENG, S.; FAN, J. Mr-dbscan: An efficient parallel density-based clustering algorithm using mapreduce. In: **Parallel and Distributed Systems, 2011 IEEE 17th International Conference on**. [S.l.: s.n.], 2011. p. 473–480.

HUANG, Y.; CHEN, C.; DONG, P. Modeling herds and their evolvements from trajectory data. In: **Geographic Information Science**. [S.l.]: Springer, 2008. p. 90–105.

HWANG, J.-R.; KANG, H.-Y.; LI, K.-J. Spatio-temporal similarity analysis between trajectories on road networks. In: SPRINGER. **International Conference on Conceptual Modeling**. [S.l.], 2005. p. 280–289.

JAGADISH, H. V. Linear clustering of objects with multiple attributes. In: **SIGMOD**. [S.l.: s.n.], 1990. p. 332–342.

JENSEN, C. S.; LIN, D.; OOI, B.-C. Continuous clustering of moving objects. **IEEE TKDE**, v. 19, 2007.

JEUNG, H.; SHEN, H. T.; ZHOU, X. Convoy queries in spatio-temporal databases. In: IEEE. **2008 IEEE 24th International Conference on Data Engineering**. [S.l.], 2008. p. 1457–1459.

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.

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.

JIANG, C.; COENEN, F.; ZITO, M. A survey of frequent subgraph mining algorithms. **The Knowledge Engineering Review**, Cambridge Univ Press, v. 28, n. 01, p. 75–105, 2013.

KAMAR, E.; HORVITZ, E. Collaboration and shared plans in the open world: Studies of ridesharing. In: **IJCAI**. [S.l.: s.n.], 2009. v. 9, p. 187.

KELLARIS, G.; PELEKIS, N.; THEODORIDIS, Y. Map-matched trajectory compression. **Journal of Systems and Software**, Elsevier, v. 86, n. 6, p. 1566–1579, 2013.

KHARRAT, A.; POPA, I. S.; ZEITOUNI, K.; FAIZ, S. Clustering algorithm for network constraint trajectories. In: **Headway in Spatial Data Handling**. [S.l.]: Springer, 2008. p. 631–647.

LEE, J.; HAN, J.; LI, X. A unifying framework of mining trajectory patterns of various temporal tightness. In: . [S.l.: s.n.], 2015.

LEE, J.-G.; HAN, J.; WHANG, K.-Y. Trajectory clustering: a partition-and-group framework. In: **SIGMOD**. [S.l.: s.n.], 2007. p. 593–604.

LEE, P.; LAKSHMANAN, L. V.; MILIOS, E. E. Incremental cluster evolution tracking from highly dynamic network data. In: IEEE. **Data Engineering (ICDE), 2014 IEEE 30th International Conference on**. [S.l.], 2014. p. 3–14.

LI, X.; CEIKUTE, V.; JENSEN, C. S.; TAN, K.-L. Effective online group discovery in trajectory databases. **TKDE**, v. 25, p. 2752–2766, 2013.

LI, X.; HAN, J.; LEE, J.-G.; GONZALEZ, H. Traffic density-based discovery of hot routes in road networks. In: **SSTD**. [S.l.: s.n.], 2007. p. 441–459.

LI, Y.; HAN, J.; YANG, J. Clustering moving objects. In: **SIGKDD**. [S.l.: s.n.], 2004. p. 617–622.

LI, Z.; DING, B.; HAN, J.; KAYS, R. Swarm: Mining relaxed temporal moving object clusters. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 3, n. 1-2, p. 723–734, 2010.

LI, Z.; LEE, J.-G.; LI, X.; HAN, J. Incremental clustering for trajectories. In: **DASFAA**. [S.l.: s.n.], 2010. p. 32–46.

LIU, H.; SCHNEIDER, M. Querying moving objects with uncertainty in spatio-temporal databases. In: SPRINGER. **Database Systems for Advanced Applications**. [S.l.], 2011. p. 357–371.

MACQUEEN, J. B. Some methods for classification and analysis of multivariate observations. In: **Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability**. [S.l.: s.n.], 1967. v. 1, p. 281–297.

MAGALHÃES, R. P.; COUTINHO, G.; MACÊDO, J.; FERREIRA, C.; CRUZ, L.; NASCIMENTO, M. Graphast: an extensible framework for building applications on time-dependent networks. In: ACM. **Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems**. [S.l.], 2015. p. 93.

MAO, J.; SONG, Q.; JIN, C.; ZHANG, Z.; ZHOU, A. Tscluwin: trajectory stream clustering over sliding window. In: **DASFAA**. [S.l.: s.n.], 2016. p. 133–148.

MERATNIA, N.; ROLF, A. Spatiotemporal compression techniques for moving point objects. In: SPRINGER. **International Conference on Extending Database Technology**. [S.l.], 2004. p. 765–782.

NANNI, M.; PEDRESCHI, D. Time-focused clustering of trajectories of moving objects. **JIIS**, v. 27, p. 267–289, 2006.

NASCIMENTO, M. A.; SILVA, J. R.; THEODORIDIS, Y. Evaluation of access structures for discretely moving points. In: SPRINGER. **Spatio-Temporal Database Management**. [S.l.], 1999. p. 171–189.

PANAGIOTAKIS, C.; PELEKIS, N.; KOPANAKIS, I.; RAMASSO, E.; THEODORIDIS, Y. Segmentation and sampling of moving object trajectories based on representativeness. **TKDE**, p. 1328–1343, 2012.

PELEKIS, N.; KOPANAKIS, I.; KOTSIFAKOS, E. E.; FRENTZOS, E.; THEODORIDIS, Y. Clustering uncertain trajectories. **KAIS**, v. 28, p. 117–147, 2011.

PELEKIS, N.; SIDERIDIS, S.; TAMPAKIS, P.; THEODORIDIS, Y. Simulating our lifesteps by example. **ACM Transactions on Spatial Algorithms and Systems (TSAS)**, ACM, v. 2, n. 3, p. 11, 2016.

PELEKIS, N.; SIDERIDIS, S.; THEODORIDIS, Y. Hermessem: A semantic-aware framework for the management and analysis of our lifesteps. In: IEEE. **Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on**. [S.l.], 2015. p. 1–10.

PELEKIS, N.; THEODORIDIS, Y.; JANSSENS, D. On the management and analysis of our lifesteps. **ACM SIGKDD Explorations Newsletter**, ACM, v. 15, n. 1, p. 23–32, 2014.

PFOSER, D.; JENSEN, C. S.; THEODORIDIS, Y. *et al.* Novel approaches to the indexing of moving object trajectories. In: **Proceedings of VLDB**. [S.l.: s.n.], 2000. p. 395–406.

POPA, I. S.; ZEITOUNI, K.; ORIA, V.; BARTH, D.; VIAL, S. Parinet: A tunable access method for in-network trajectories. In: IEEE. **2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)**. [S.l.], 2010. p. 177–188.

POPA, I. S.; ZEITOUNI, K.; ORIA, V.; BARTH, D.; VIAL, S. Indexing in-network trajectory flows. **The VLDB Journal—The International Journal on Very Large Data Bases**, Springer-Verlag New York, Inc., v. 20, n. 5, p. 643–669, 2011.

POPA, I. S.; ZEITOUNI, K.; ORIA, V.; KHARRAT, A. Spatio-temporal compression of trajectories in road networks. **GeoInformatica**, Springer, v. 19, n. 1, p. 117–145, 2015.

POTAMIAS, M.; PATROUMPAS, K.; SELLIS, T. Sampling trajectory streams with spatiotemporal criteria. In: IEEE. **18th International Conference on Scientific and Statistical Database Management (SSDBM'06)**. [S.l.], 2006. p. 275–284.

ROH, G.-P.; HWANG, S.-w. Nncluster: An efficient clustering algorithm for road network trajectories. In: SPRINGER. **International Conference on Database Systems for Advanced Applications**. [S.l.], 2010. p. 47–61.

SILVA, J. A.; FARIA, E. R.; BARROS, R. C.; HRUSCHKA, E. R.; CARVALHO, A. C. de; GAMA, J. Data stream clustering: A survey. **ACM Computing Surveys (CSUR)**, ACM, v. 46, n. 1, p. 13, 2013.

SILVA, T. Coelho da; ARAUJO, A. C. N.; MAGALHAES, R. P.; FARIAS, V. A. E. d.; MACEDO, J. A. de; MACHADO, J. C. Efficient and distributed dbscan algorithm using mapreduce to detect density areas on traffic data. In: . [S.l.]: ICEIS, 2014.

SILVA, T. Coelho da; MACÊDO, J. A. de; CASANOVA, M. A. Discovering frequent mobility patterns on moving object data. In: ACM. **Proceedings of the Third ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems**. [S.l.], 2014. p. 60–67.

SILVA, T. Coelho da; ZEITOUNI, K.; MACÊDO, J. A. F. de; CASANOVA, M. A. On-line mobility pattern discovering using trajectory data. In: **EDBT**. [S.l.: s.n.], 2016. p. 682–683.

SILVA, T. Coelho da; ZEITOUNI, K.; MACÊDO, J. de. Online clustering of trajectory data streams. In: **MDM**. [S.l.: s.n.], 2016.

SILVA, T. Coelho da; ZEITOUNI, K.; MACÊDO, J. de; CASANOVA, M. A. Cutis*: optimized online clustering of trajectory data stream. In: **IDEAS**. [S.l.: s.n.], 2016.

SILVA, T. Coelho da; ZEITOUNI, K.; MACÊDO, J. de; CASANOVA, M. A. A framework for online mobility pattern discovery from trajectory data stream. In: **MDM**. [S.l.: s.n.], 2016.

SPILIOPOULOU, M.; NTOUTSI, I.; THEODORIDIS, Y.; SCHULT, R. Monic: Modeling and monitoring cluster transitions. In: **Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2006. p. 706–711.

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: **ICDE**. [S.l.: s.n.], 2012. p. 186–197.

TANG, L.-A.; ZHENG, Y.; YUAN, J.; HAN, J.; LEUNG, A.; PENG, W.-C.; PORTA, T. L. A framework of traveling companion discovery on trajectory data streams. **ACM Transactions on Intelligent Systems and Technology (TIST)**, ACM, v. 5, n. 1, p. 3, 2013.

THAT, T.; HAI, D. *et al.* **Gestion efficace et partage sécurisé des traces de mobilité**. Tese (Doutorado) — Paris Saclay, 2016.

THEODORIDIS, Y.; VAZIRGIANNIS, M.; SELLIS, T. Spatio-temporal indexing for large multimedia applications. In: IEEE. **Multimedia Computing and Systems, 1996., Proceedings of the Third IEEE International Conference on**. [S.l.], 1996. p. 441–448.

TIAKAS, E.; PAPADOPOULOS, A.; NANOPOULOS, A.; MANOLOPOULOS, Y.; STOJANOVIC, D.; DJORDJEVIC-KAJAN, S. Searching for similar trajectories in spatial networks. **Journal of Systems and Software**, Elsevier, v. 82, n. 5, p. 772–788, 2009.

TIAKAS, E.; PAPADOPOULOS, A. N.; NANOPOULOS, A.; MANOLOPOULOS, Y.; STOJANOVIC, D.; DJORDJEVIC-KAJAN, S. Trajectory similarity search in spatial networks. In: IEEE. **2006 10th International Database Engineering and Applications Symposium (IDEAS'06)**. [S.l.], 2006. p. 185–192.

TZOURAMANIS, T.; VASSILAKOPOULOS, M.; MANOLOPOULOS, Y. Overlapping linear quadtrees: a spatio-temporal access method. In: ACM. **Proceedings of the 6th ACM international symposium on Advances in geographic information systems**. [S.l.], 1998. p. 1–7.

VIEIRA, M. R.; BAKALOV, P.; TSOTRAS, V. J. On-line discovery of flock patterns in spatio-temporal data. In: ACM. **Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems**. [S.l.], 2009. p. 286–295.

VLACHOS, M.; KOLLIOS, G.; GUNOPULOS, D. Discovering similar multidimensional trajectories. In: **ICDE**. [S.l.: s.n.], 2002. p. 673–684.

XU, K. S.; KLIGER, M.; III, A. O. H. Adaptive evolutionary clustering. **Data Mining and Knowledge Discovery**, Springer, v. 28, n. 2, p. 304–336, 2014.

YIU, M. L.; MAMOULIS, N. Clustering objects on a spatial network. In: **Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data**. [S.l.: s.n.], 2004. p. 443–454.

YU, Y.; WANG, Q.; WANG, X.; WANG, H.; HE, J. Online clustering for trajectory data stream of moving objects. **ComSIS**, p. 1293–1317, 2013.

YUAN, J.; ZHENG, Y.; XIE, X.; SUN, G. Driving with knowledge from the physical world. In: ACM. **Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2011. p. 316–324.

YUAN, J.; ZHENG, Y.; ZHANG, C.; XIE, W.; XIE, X.; SUN, G.; HUANG, Y. T-drive: driving directions based on taxi trajectories. In: ACM. **Proceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems**. [S.l.], 2010. p. 99–108.

ZHANG, T.; RAMAKRISHNAN, R.; LIVNY, M. Birch: an efficient data clustering method for very large databases. In: ACM. **ACM SIGMOD Record**. [S.l.], 1996. v. 25, n. 2, p. 103–114.

ZHENG, K.; ZHENG, Y.; YUAN, N. J.; SHANG, S. On discovery of gathering patterns from trajectories. In: **ICDE**. [S.l.: s.n.], 2013. p. 242–253.

ZHENG, Y. Trajectory data mining: an overview. **TIST**, p. 29, 2015.