UNIVERSITÀ DI PISA

Efficient Algorithms for Time-Based Similarity for Trajectories on Graphs

Author:
Shima Moghtasedi

Supervisors:
Roberto Grossi
Andrea Marino

March, 2020

Abstract

The analysis of similar trajectories in a network provides useful information for route recommendation or fraud detection. In this thesis, we are interested in algorithms to efficiently retrieve similar trajectories. Many studies have focused on retrieving similar trajectories by extracting the geometrical information of trajectories. We provide a similarity function by making use of both the temporal aspect of trajectories and the structure of the underlying network. We propose exact and approximation techniques that offer the top-k most similar trajectories with respect to a query trajectory within a given time interval in an efficient way. We also investigate how our ideas can be applied to similar behavior of the tourists, so as to offer a high-quality prediction of their next movements.

Acknowledgements

I am very fortunate to have performed my Ph.D. at the University of Pisa, which has been a truly life-changing experience for me and it would not have been possible to do without the support and guidance that I received from many people.

I would like to first express my sincere gratitude to my thesis advisor Prof. Roberto Grossi for all the support and encouragement he gave me, for his patience and immense knowledge. Many thanks to my second advisor Prof. Andrea Marino for the guidance, encouragement, and advice he has provided throughout my research. Without their guidance and constant feedback, this Ph.D. would not have been achievable.

I would like to thank my thesis committee: Prof. Angelo Maurizio Bonuccelli and Prof.Anna Monreale, for their insightful comments and encouragement. I appreciate the discussion times I have had with Prof. Anna Monreale.

My sincere thanks also go to Prof. Apostolos N. Papadopoulos, who believed in my research and provided me an opportunity to join their team at the University of Thessaloniki. I would like to thank Prof. Elefteris Tiakas for his insightful comments and for sharing with me his experience in the trajectory analysis field. Without their precious support, it would not be possible to conduct this research.

I greatly appreciate the support received through the collaborative work undertaken with Prof. Franco Maria Nardini and Prof. Cristina Ioana Muntean at ISTI–CNR, Italy.

I am grateful to my friends and colleagues in the Universities of Pisa and Thessaloníki for making a warm environment of working. I especially thank Ioanna Miliou and Mahsa Kohandel for always being there for me in hard moments of the Ph.D. journey.

Last but not the least, my deep and sincere gratitude to my family for their continuous love, help, and support. I am forever indebted to my parents for giving me the opportunities and experiences that have made me who I am. Thanks to my brother for always being there for me as a friend. I am also grateful to my departed grandmother for her valuable prayers. Specially and finally, I would like to thank my husband, who has been a constant source of support, encouragement, and patience during the challenges of my Ph.D. life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Due to the development of location-based devices such as GPS and some social networks that provide the service of route sharing and digital traces of many human activities such as web browsing, a large amount of data recording the positions of moving objects, called trajectory, are generated and collected. The efficient analysis of trajectory data fleet management applications analyze trajectories of vehicles towards route optimization [42], aiming at more efficient and effective resource usage. Knowledge extracted from trajectory data can be used in a smart city environment to detect and visualize novel or anomalous events [58]. Finally, trajectories that have visited specific points of interest may be used as a guideline for others that wish to visit the same or similar places [51].

This made researchers analyzing the motion patterns of moving objects. The trajectory similarity search (i.e. retrieving the similar trajectories with respect to a given query argument such as point(s) and trajectory) has been an attractive and challenging topic due to the wide range of the applications taking benefit of similarity between moving objects, such as the following real and practical applications:

(I) Tourism applications: given the recent movement of a tourist and the current time, some application may be interested in predicting next location of the tourist taking advantage of the similarity between the behavior of tourists, because of the marketing reason, or providing a high-quality trip planning for the tourist.

(II) Fraud detection: given the trajectories of all taxi drivers, the authorities can detect the taxi driving frauds by providing a trajectory evidence [34].

(III) Infection detection: as recently the health authorities in the world have been monitoring the spreading coronavirus among the individuals. May one approach to reducing the spreading speed of this new virus be finding the individuals that may have the risk of infection. The individuals that have been close to the cases that they have newly detected, spatially and temporally, probably have a high risk to be affected in the future. Based on this basic idea, we can analyze the mobility of the affected individuals obtained from, for example, the mobile call records and the check-in airport dataset, to make the trajectory set. Thereafter, the most similar trajectories to the trajectory of the affected case make a set of high-risk individuals, which need to be separated in quarantine.

(IV) Traffic analysis: By identifying the similar trajectories, effective data mining techniques (e.g., clustering) can be applied to discover traffic jams or predict traffic in some area of a city.

Therefore, this thesis is motivated to focus on studying the behavior of trajectories that are defined on a graph, annotating with temporal data.

1.1 Trajectory Representation

The majority of existing methods for trajectory similarity represent trajectories as a sequence of two-dimensional GPS points across time and embedded on three-dimensional space (x, y, z). However, in many real-life scenarios, trajectories are constrained by the underlying network. This means that objects cannot move freely but they must obey the rules applied by the network. Therefore, each trajectory may be represented by a sequence of nodes.

Evidently, moving from node u to node v requires the existence of a link (edge) between u and v. For example, in a road network, nodes may correspond to junctions, points of interest (PoI) or user-defined locations in a map, and edges may correspond to road segments. As an example, consider the mobile telephone network, which is usually supported by a large set of base stations. The motion from one area to another is reflected by the change of the base station that the mobile phone is connected to. In this scenario, nodes represent base stations and edges denote neighboring base stations. In these cases, the GPS trajectories are transferred to a *spatial network*, which is a network that is modeled as a graph with the set of nodes embedding on two-dimensional space (e.g. road network). Therefore, a trajectory is a sequence of nodes on a spatial network across time. This kind of trajectories are called spatiotemporal trajectories.

Having a large-scale collection of spatiotemporal trajectories raises the challenge of data management and retrieving relevant data efficiently. Moreover, there are other cases that annotating the nodes of the network by the location information is either not feasible or not meaningful. For example, consider the Web graph, which represents the network of URLs. Each node represents a Web page, and each directed edge represents the link (out-link) from node u to node v. The time spent by each user to a page is also recorded, and it is used in expressing trajectory similarity. Although the annotation of a web page with location-based information is feasible [7], it cannot be used in a meaningful way of studying the similarity between the movement of the different users between URLs. We can mention the Internet network as another instance of this kind of network, in which the routers make the nodes of the graph and the connection between routers in the network, make the edge set of the graph. In this kind of network, may someone be interested in studying the movement of the packets that are transferring between routers.

In this thesis, we aim at representing the movements by using the *topology* of the networks, in which the network is modeled as a graph with no spatial information, to reduce the dimensionality of the data. Briefly, the idea is to reduce movements to occur in networks without spatial information. As a consequence, the movement data becomes a sequence of nodes on a graph. The advantages of considering such lower-dimensional trajectories are the reduced the overall size of the data and the lower-dimensional indexing challenges. Moreover, the ways of analyzing this kind of trajectories are more scalable, since they can apply on both GPS trajectories and other kind of trajectories. This is due to that, it is possible to build a graph based on the trajectories and mine the relationships among trajectories with respect to the algorithms on graphs.

In some applications the temporal information of trajectories plays the most important role to specify the similarity between trajectories. For example, the moving paths of athletes on tracks are always the same while the difference in these trajectories is their time information. Therefore, each trajectory may be represented by a sequence of nodes annotated by the corresponding timestamps (or time intervals).

1.2 Similarity Measure

Most of the existing methods for trajectory similarity assume that trajectories are allowed to move freely on the space, without motion restriction and therefore use the Euclidean distance between two trajectories to mea-

sure the similarity of trajectories [92, 4, 85, 23, 80]. Taking into account the network can lead to specific models that reflect the similarity of trajectories on the network, that is two trajectories that are similar regarding Euclidean distance may be dissimilar considering network distance. Moreover, the temporal aspect of a trajectory represents the movement varying with time and only considering the spatial aspect of trajectories and ignoring the temporal information is an obvious limitation in discovering the similarity of trajectories. Therefore, a similarity function must take into account the temporal information of trajectories.

On the top of the spatiotemporal similarity functions that are mostly a linear combination of the independently computed spatial and temporal distances [81, 79, 76, 82], this project takes into account both the temporal information of trajectories constrained on a network and their location on the network, in a single function, namely called structural-temporal similarity function. As stated in [41] it is difficult to define the similarity between trajectories by spatiotemporal distance directly. Here, we will compute the similarity between two trajectories, temporally and topologically, using a similarity function. We aim at exploiting the topology of the network, assessing that two trajectories are similar if they pass through closeby nodes at roughly the same time.

Indeed, the time complexity of a spatiotemporal similarity measure is an important criterion. A typical similarity trajectory model compares a huge number of trajectories, in which each trajectory may have a large number of nodes. Determining the most similar trajectories to a given query trajectory from a dataset may require a large number of comparisons. In particular, considering two trajectories, a similarity function at each comparison between these two trajectories, computes a distance between either two nodes, if the trajectories are constrained on a network, or two sample points if trajectories are on space. Computing the network distance between two nodes is a big challenge when it is supposed to be used in a similarity function between a huge number of trajectories. Most existing measures require quadratic time [1] for similarity computations, as shown in Table 2.3. There are some studies proposing the similarity functions that require linear time to be computed but they have limitations (see the Chapter 2 for more details). As far as we know, there is no linear-time similarity that considers trajectories with any arbitrary lengths and with a flexible notion of proximity in time and location

[1] We measure the time complexity of each similarity function in terms of the number of distances needed to compute by each function.

on the network. In this thesis, we show it is possible to consider both the time and structure of the trajectories (i.e. the location of the trajectories on a graph) to overcome the limitation of the current literature, and a good compromise can be found. We refer the reader to Chapter 2 for a comparison between the similarity functions in the literature.

1.3 Trajectory Management

Trajectory retrieval is arguably the most important application for trajectory similarity queries, that is, given a query trajectory retrieving the most similar trajectories to the query with respect to a similarity measure. The most representative scenario is the k-Nearest Neighbor queries (k-NN), which aims to determine top-k most similar trajectories to a given query trajectory. When extracting relevant information querying large trajectory datasets, the performance crucially depends upon an efficient index of trajectories that can cluster nearby trajectories together and helps to prune irrelevant trajectories. The majority of the existing methods for trajectory similarity on the networks are based on the R-Tree [1]. These have been designed for trajectories that are mapped to a spatial network (e.g. road network). These structures are not suitable for efficiently retrieving the trajectories on a graph with no spatial information. However, the works in [54, 69] provide indexing for spatiotemporal trajectories on the graph, which index the time instances of trajectories arriving in a node. These structures are not suitable, concerning the time intervals that the trajectories spend in the nodes (more discussions provided in Chapter 2).

When faced with a new type of data as the network constrained trajectories, how to efficiently organize this data to process queries against this data is an important challenge. As indexing methods typically designed for trajectories are already mapped to a spatial network, and none of them consider the time intervals the objects spend in the nodes of the networks, using existing access methods considering trajectories are defined in this project, is not either possible or efficient. In this thesis, we attempt to propose an indexing structure based on the *interval tree* structure [25] to store trajectories on the graph. We model our structure in the main memory to support short response time, although it is possible to extend the methods considering external memory interval tree [6, 27].

1.4 Similarity Query Processing

Studying the similarity between trajectories on networks, the query could be a single node, multiple nodes, and a trajectory. These queries aim at identifying the trajectories that are most similar to the given query (i.e. node(s), trajectory) with respect to a similarity function. In this thesis, we study the general case, that is, given a trajectory on the graph, we aim at determining those trajectories are most similar to the given query trajectory. Since computing the network distance between trajectories is expensive, a query processing algorithm typically aims at minimizing the number of distance computations. In one approach a query processing algorithm employs a prune-and-refine approach with the aim to reduce the number of total similarity score computations, which consequently, reduce the number of distance computations [23, 80, 79]. The *prune* step finds a set of candidate trajectories that are likely to be the results. The *refine* step is to further identify the exact query result from the set of candidates. The overall efficiency of a query processing algorithm mainly depends on the effectiveness of the pruning step. Another approach is providing an approximate searching with the aim to reduce the number of distance computation [3].

Regarding the first approach, Chen et al. [23] propose the method based on the searching by location, in which for a given set of query locations, they search for nearby trajectories for each query location, separately. Then the intersection of the results contains the top-k trajectories that are close to all query locations. By following the same strategy Tang et al. [80] retrieve top-k trajectories with minimum aggregated distance to a set of query locations. While, these top-k most similar searching process defined on spatial trajectories on Euclidean space and a spatial index (e.g., R-tree [37]) is used to enhance the query efficiency. Consequently, their similarity functions only take spatial proximity into account. Instead, we involve the temporal aspect of trajectories to measure the similarity between trajectories. By taking into account the spatiotemporal similarity, the PTM query [79] searches by each node of query for close trajectories in the spatial and temporal domain, independently. The PTM uses the network expansion [26] for finding trajectories that are spatially close to each node of the query. Then, by considering all timestamps of trajectory nodes on a time axis, they find those trajectories have a timestamp within a predefined range of each query timestamp.

Regarding the second approach, [3] propose an approximate method to retrieve k-NN trajectories to the set of query points on the plane, by considering the centroid of the convex hull of all query points, as a single query point.

1.5 Problem Statement

In this thesis, we propose a time-based most similar trajectory query, where a query is a sequence of nodes of a topological network with the corresponding time intervals and a given query time interval, aiming at discovering top-k trajectories that are most similar to the query within the query time with respect to the structural-temporal similarity function, called K-MSTRAJ . For example, a query with two nodes u, v on a topological network appears as:

```
< (u, [1307114479, 1307115648]), (v, [1307115649, 1307115649])  >
```

where $[1307114479, 1307115648]$ is the time interval in ms the query trajectory spent on the node u. Consequently, we need to measure the distance between trajectories in the dataset and the query within the time intervals. We need to manage the trajectories without any spatial information of the underlying network.

K-MSTRAJ

In this thesis, we aim at addressing K-MSTRAJ query in an efficient way. A straightforward approach to process K-MSTRAJ is to compute the similarity score for each trajectory in the dataset and report the k trajectory ids with maximum scores within the query time interval. This approach is inefficient with respect to the number of distance computations. To accelerate the computations, we provide two kind of methods:

- Prune-and-refine: In particular, we provide a technique to efficiently produce an exact solution to the query, by computing the minimum number of similarity score computations. Moreover, we also aim at using this technique to design a fast and accurate approximate method.

- Shrink-based: We provide a brief and comprehensive representation of each trajectory in the dataset by using network Voronoi partitioning with the aim to reduce the number of shortest path distance computations.

Next-PoI Prediction

As mentioned earlier, analyzing the similarity between trajectories provides useful knowledge in a wide range of applications such as tourist applications. In this direction, one of the fundamental problems in tourist trajectory mining is the next location prediction, that is, predicting the most likely "*Point of Interest*" (PoI) to be visited by a tourist during its tour of a given city.

In this thesis, we aim at identifying the PoI that a tourist will visit in the future with the highest probability. The objective of this study is to reflect the similar behavior of the past tourists by considering temporal information of trajectories, which is really useful in many applications, to predict the next movement of a tourist.

1.6 Contribution and Thesis Organization

Our research is motivated by two requirements. First, our method should be based on the characteristics of moving objects on networks and not necessarily a spatial network. Second, we should simultaneously consider a structural-temporal similarity. Based on these ideas, we propose a search method for finding trajectories on networks from the trajectory database similar to a given query trajectory within a given time interval. In this thesis, our main contributions can be summarized as follows: (1) we propose a low complexity similarity function considering temporal aspect and the location of trajectories on the graph, directly in a single function; (2) we propose a storage scheme to speed up searching for similar trajectories; (3) we use the Network Voronoi Diagram to process the query by computing the minimum number of shortest path distances; (4) we provide the fast exact and approximate prune-and-refine techniques to process query by the minimum number of similarity score computations; (5) we propose a technique to predict the next movement of a user in tourist application, by taking into account the structural-temporal similarities between tourists and without involving machine learning techniques, which is the basic approach for addressing this problem.

These contributions are organized in some manuscripts as follows.

1. "Time-Based Similar Trajectories on Graphs"; Published in Proceedings of the 19th Italian Conference on Theoretical Computer Science, 2018, pp. 82-86 (2018).

2. "Finding Structurally and Temporally Similar Trajectories in Graphs"; Accepted in the SEA conference (2020).

3. "Efficient Top-k Similarity Query Processing in Network-Constrained Trajectories"; To be submitted in the Information system journal.

4. "Next Movement Prediction of Tourists"; To be submitted in the ASONAM conference (2020).

The rest of this thesis is organized as follows. Chapter 2 reviews related works. In Chapter 3 we present some basic concepts and definitions that are necessary to develop the techniques in this thesis. We explain the main problem that this thesis focuses on it. Thereafter, we propose an access method and due to the lack of competitors we design a baseline method for solving the proposed problem to compare with. We evaluate the provided strategies in comparison with this baseline method. In Chapter 4 we propose an approximation technique based on the network Voronoi diagram that offers the top-k most similar trajectories with respect to a query trajectory in an efficient way with acceptable precision. We investigate our method over real-world networks. The performance evaluation results show the effectiveness of the proposed method. In Chapter 5 we propose some fast prune-and-refine techniques to accelerate the query processing, by providing the exact and approximate top-k most similar trajectories to the query. We conduct comprehensive experiments to evaluate the effectiveness of the proposed methods in this chapter. In Chapter 6, we focus on the problem of predicting the next movement of a tourist and experimentally compare the performance of the proposed prediction model with state-of-the-art methods. Finally, we conclude the project and discuss the future direction of our studies in Chapter 7.

Chapter 2

State of the Art

2.1 Modeling Moving Objects(Moving Objects Representation)

With the rapid growth of wireless communications and location-based devices, the concept of Moving Object Databases (MOD) has become important in the spatial and spatiotemporal database research. Moving objects dataset contains a set of objects whose positions change with respect to time. The position of moving objects while they are moving along the time could have useful knowledge for different applications like route optimization or recommendation. In this section, we classify the movement of objects into two major types *constrained* and *un-constrained*, with respect to the environment they move on and mentioning some applications of each type.

2.1.1 Trajectory

In order to record the movement of an object, we would have to know the position of the object at all times, i.e., on a continuous basis. The data obtained from moving objects is similar to a *string*, arbitrary oriented in 3D space, where two dimensions correspond to space and one dimension corresponds to time. We can have the movement of an object by sampling the movement. There are some positioning technologies like Global Positioning System (GPS) that can be used to sample the location of moving objects at discrete instances of time, such as every few seconds. By sampling the movement of an object, we obtain a sequence of triples $\langle (x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_N, y_N, t_N) \rangle$, representing the *trajectory* of the moving object: x_i, y_i represent the geographic coordinates of the moving object at time t_i, which is called *raw trajectory* and provides a *point-based tra-*

jectory representation. Representing the movement of objects as a sequence of time-stamped points is a popular and simple way of modeling the object's movements.

In this kind of trajectories, we cannot get some information at times in-between sampled points. Therefore, in some applications which are needed to obtain the entire movement, they need to use an interpolation operation: a trajectory is represented as a sequence of connected segments in which each of them connects two consecutive sampled points. The linear interpolation is widely used to estimate the location of the moving object between sampled points. It provides a *segment-based trajectory representation* of the movement of objects, which a trajectory is a sequence of the connected segments [62, 71, 68, 67]. In these trajectories, each segment is represented by two consecutive sampled points. The traces of the movements of people or animals [16, 36] which can move freely on the plane or space, can provide this kind of trajectories.

Representing the object's movements as a sequence of time-stamped geographical coordinates costs a lot of overhead for communication, computation, and data storage. Indeed, in several applications, the movement is restricted physically and spatially. For example, a large number of GPS equipped vehicles such as taxis, cars, buses, airplanes move on a pre-defined *spatial network* like roads, railways, air routes [67]. Considering this kind of moving objects, some applications may be interested only in the position of objects with respect to the network, rather than in their exact coordinates. In these applications, the trajectory of moving objects can be represented as the component of the network, as a sequence of edges or nodes of the underlying network. In particular, the sample points of the raw trajectories would be mapped to the nodes (edges) on a spatial network by using the map-matching algorithms [13, 49]. The spatial network is a graph defined as follows.

Spatial Network Generally, networks are modeled as a graph $G(V, E)$, where V denotes the set of nodes and E denotes the set of edges of the network. Each edge between two nodes $u, v \in V$ shows a connection between u, v. Regarding the purpose of the graph creation, the edge could be weighted or directed. In particular, a spatial network is a network with the additional spatial information assigned to the components of the graph.

A *road network* is an example of a spatial network, which could be represented as a weighted graph $G(V, E)$, such that each node represents a road junction, and each edge represents a road segment. Each node of the graph is assigned to the spatial position in terms of geographical coordinates (i.e.,

latitude and longitude). The weight of an edge between two nodes $u, v \in V$ of the road network usually is the Euclidean distance between two spatial coordinates assigned to u, v. There are two representations for the road network: a *geometric* view and a *topological* view. The geometric view (or 2D view) captures the geographic locations of the road network components. This is the base view of the road network. The topological view uses a graph in order to represent the road sections and the intersections with no spatial information.

For instance, by mapping the sample points of a raw trajectory onto the nodes of a network, we obtain a sequence of pairs $\langle (v_1, t_1), (v_2, t_2), ..., (v_n, t_N) \rangle$ in which each $v_i \in V$ is assigned to the pair (x_i, y_i) of its position in Euclidean space and t_i is the timestamps of the sample point (x_i, y_i).

Moreover, there are some other types of object's movements that cannot be considered to be physically constrained or spatially bounded, like Email communication or Web browsing [7, 84, 10]. Due do that there is no spatial dimension for this kind of trajectories, the strategies proposed in the literature for studying the behavior of trajectories do not work on them. For example, in [10] for studying the web browsing mobility, by defining a sequence of visited web pages by users, first, they create a graph of trajectories. Then, they use the Fruchterman-Reingold graph layout algorithm [33] to transfer the created graph onto a 2D space, for applying the spatial-based algorithm.

Table 2.1 shows two types of trajectories: constrained and un-constrained trajectories and mentions a few applications in each type.

2.2 Trajectory Query Types

Trajectory search queries aim at finding the most relevant trajectories to the query arguments, which could be single or multiple point(s), a region, and a trajectory [23, 78, 79, 98, 97]. The relevancy between a trajectory and a query argument may contain spatial [23], spatial-temporal [81, 79], spatial-textual [78, 98, 98], and density elements [75].

Therefore, we generally classify the trajectory queries into three types:

1. Region-based query (Range query): looking for trajectories passing through a given spatiotemporal region.

2. Point(s)-based query: looking for the trajectories which satisfy the query requirements to a specified query point(s).

3. Trajectory-based query: looking for trajectories are similar (in terms of the query requirements) to the given trajectory.

Movement	Moving object	Network	Application
Un-constrained movement	Animal	×	✓ Mining migration pattern ✓ Studying the distribution and habitat of animals [46, 48]
	Human: users of mobile phones or users of location based social networks like Flickr, Instagram	×	✓ Studying the behavior of the users based on the sequence of cell phone towers that the user made a call [65], ✓ Trip recommendation based of the sequence of the time-ordered of geo-taged photos [60]
Network constrained movement	Real objects: taxis, cars, buses, airplanes, pedestrians on road networks and transportation networks	Spatial Network	✓ Traffic control, ✓ Scheduling transportation ✓ Internet marketing [67, 68, 39]
	Individual objects: packets on routing networks or computer networks, click streaming on web networks	Non-Spatial Network	✓ Predicting a user's access on website, ✓ Studying the pattern behavior of users [35, 10]

Table 2.1: Diversion of the object's movements w.r.t the environment the objects are moving on.

Considering the constrained trajectories, a point may be a node of the network and a region could be a set of nodes.

Region-based queries The range queries retrieve the trajectories that have an intersection with a given spatial and temporal range. In [75], trajectory search by region (TSR) retrieves trajectories with the highest spatial-density correlation to a set or sequence of query regions. On the other hand, a query may be a query point and a spatial radius, which aims at finding the trajectories traverse the region within the given radius from the query point as studied in [50].

Point(s)-based queries The single point based query [68, 31] looks for the closet trajectories to only one location (e.g. the supermarket in a city, the food source to animals). In addition, a query point can be annotated to textual attribute like the spatial-textual query in [97], which looks for trajectories are spatially close to the query point and also satisfy the semantic requirements defined by the query.

The set of point query [23], finds the trajectories are close to a set of specified query points (e.g., sightseeing places). This query is useful for trip recommendation to travelers that they desire to visit some specified points of interest.

Having query points annotated by some textual preferences, the work in

[78] finds the trajectories are spatially close to all query points and its textual attributes are similar to the given one.

Trajectory-based queries An interesting type of query that is useful in many applications is the so-called trajectory similarity problem, which aims at finding *similar* trajectories of moving objects with respect to a similarity function. We classify the trajectory-based queries into two kinds of queries: Boolean similar trajectories search and top-k similar trajectories search query.

1. Boolean similar trajectories: finds all trajectories that are similar to a given query trajectory [40, 41].

2. Top-k similar trajectories: retrieves trajectories that are most similar to the given query trajectory w.r.t a spatiotemporal similarity [20, 21, 79, 81] or a spatial-textual similarity [98] functions.

We classify the similarity functions into two categories: spatial functions, which they only consider the spatial information of trajectories, and spatiotemporal functions, which consider both temporal and spatial aspects of trajectories. This section will introduce the spatial and spatiotemporal similarity measures that measure the similarity between two trajectories by considering both the spatial and temporal aspects of trajectories.

2.3 Basic Similarity Measures

Many studies on similarity-based retrieval of trajectories have been conducted in the last two decades. The first and most difficult challenge in these studies is to give a good definition of similarity/distance between a trajectory and a query argument (i.e., point(s) and trajectory). In the similarity functions, d is some function distances between two points, which can be either the L_p-norm on Euclidean space or network distance on the spatial network.

2.3.1 Point(s)-Trajectory

The distance between a point q and a trajectory T is generally measured as the aggregated distance as follows:

$$dis(q, T) = f_{p_i \in T}(d(q, p_i))$$

where, $f(.)$ is a function that could be: the sum of distances, maximum distance, and minimum distance. While, most studies in the literature [82,

23] assume this function as the minimum distance, i.e. the distance from q to the nearest point of T. This pair of points is namely called the match pair points.

The work [23] extends this distance function for a single query point to a small set of query points, and proposes the following distance function between a set Q of n query points and a trajectory T by using the exponential function:

$$Sim(Q,T) = \sum_{q \in Q} e^{-dis(q,T)}$$

The intuition of using the exponential function is to assign a larger contribution to a closest matched pair of points while giving much lower value to those faraway. The trajectories can move freely on the plane and the query is specified in spatial terms only. Thus, the similarity functions are based on the spatial distance (i.e Euclidean distance) between trajectories and query points on the plane. However, this measure is sensitive to the distance unit and the query results could be different if the distance unit changes from kilometer to mile.

Therefore, the work [82, 80] computes the similarity between the set of query points Q and the trajectory T as an augmentation of the distance between each query point and the trajectory as follows:

$$Sim(Q,T) = \frac{1}{n} \sum_{q \in Q} dis(q,T)$$

2.3.2 Trajectory-Trajectory

Many distance functions have been proposed considering trajectories as sequences, i.e. like strings. Lp-Norms [4, 29] are distance measures that are suitable for trajectories having the same length. Another measure is Dynamic Time Warping, (DTW) [92, 45] which has been proposed for trajectories having (possibly) different lengths. However, in these approaches, some outlier points from a trajectory, which could be considered as noise points, may cause a big distance between trajectories. For this reason, the concept of the Longest Common Sub-Sequence (LCSS) [85] has been employed. LCSS-based distance allows for ignoring the noise points. However, it needs a matching threshold to determine whether to ignore a point of the trajectory. Edit Distance on Real Sequence (EDR) [21] is similar to LCSS in using a threshold parameter to determine if two points are matched while considering penalties to gaps. Edit distance with Real Penalty (ERP) [20] also introduces combining the merits of DTW and EDR. It is worth noting that

these similarity functions are based on the *edit distance*, which are originally proposed for matching strings. But by representing trajectories as strings, these functions can be applied to trajectories.

Pairwise Distance

Considering two trajectories Q, T as a sequence of points with the same length n, [4] simply computes the distance between two trajectories as follows:

$$\mathcal{D}(Q,T) = (\sum_{i=1}^{n} |q_i - p_i|^p)^{1/p}$$

where $q_i \in Q$ and $p_i \in T$. This function computes the Manhattan and Euclidean distance between points when $p = 1$ and $p = 2$, respectively.

We can also extend this definition to measure the distance between two trajectories that are constrained on the spatial network. Considering each trajectory as a sequence of nodes on graph in spatial domain, the works [81, 18] have studied this kind of trajectories and define the spatial distance as follows:

$$\mathcal{D}_{net}(Q,T) = \frac{1}{n} \sum_{i=1}^{n} d(v_i, u_i)$$

where $u_i \in Q$ and $v_i \in T$ and n is the length of trajectories in terms of the number of nodes.

The main drawback of these functions is that they are based on point-to-point distance calculations and require two trajectories with the same lengths, which is rarely could be happened in the real applications. Therefore, Dynamic Time Warping, is proposed for trajectories with the arbitrary length.

Dynamic Time Wrapping (DTW) [92]

Considering two trajectories with possibly different lengths, DTW uses a recursive manner to search all possible point combinations between two trajectories for the one with minimum distance, which can be computed with the dynamic programming. However, in these approaches, some outlier points from a trajectory, which could be considered as noise points, may cause a big distance between trajectories. For this reason, the concept of the Longest Common Sub-Sequence (LCSS) [85] has been employed.

Longest Common Sub-Sequence (LCSS) [85]

LCSS-based distance allows skipping the noise points when calculating the distance of trajectories, using a threshold to establish a match for two points from two different trajectories. While DTW must match all points, the LCSS allows us to skip noisy points and so match some points of trajectories rather than all of them. This idea makes LCSS more flexible to the trajectories with faraway points. Let Q and T be two trajectories of moving objects with size n and m respectively, where $Q = \langle v_1, v_2, \cdots, v_n \rangle$ and $T = \langle u_1, u_2, \cdots, u_m \rangle$. For a trajectory T, let $Head(T)$ be the sequence $Head(T) = \langle u_1, u_2, \cdots, u_{m-1} \rangle$. Given an integer σ and a real number $0 < \epsilon < 1$, the $LCSS(Q, T)$ is defined as follows:

$$LCSS(Q,T) = \begin{cases} 0 & \text{if } m = 0 \\ & \text{or } n = 0 \\ 1 + LCSS(Head(A), Head(B)) & \text{if } d(v_n, u_m) < \epsilon \\ & \text{and } |n - m| \leq \sigma \\ \max(LCSS(Head(Q), T), LCSS(Q, Head(T)) & \text{otherwise} \end{cases}$$

There are some studies that define a function based on the LCSS to measure the similarity between trajectories. The work in [79] combine the idea of the distance function proposed in [23] and LCSS function to measure the similarity between two spatial trajectories on the spatial network. Given a matching spatial threshold ϵ, it defines the spatial influence factor between two spatial points in a spatial network as follows:

$$I_s(v,u) = \begin{cases} 0 & \text{if } d(v,u) > \epsilon \\ e^{-d(v,u)} & \text{otherwise} \end{cases}$$

Then, the similarity between two trajectories Q, T is defined by the following function:

$$Sim(Q,T) = \max \begin{cases} I_s(v_n, u_m) + Sim(Head(Q), T) \\ Sim(Q, Head(T)) \end{cases}$$

Although, LCSS is robust to noises, the accuracy of the results depends on the matching thresholds. This is due to that, it does not consider some unmatched points and ignoring these points depends on the predefined threshold, which may lead to some inaccuracy.

Edit Distance Based [21, 20]

The Edit Distance on Real Sequence (EDR) [21] aims at providing a less coarse description than LCSS for the distance between trajectories. EDR removes the noise effects by quantizing the distance between a pair of elements to two values, 0 and 1. Furthermore, assigning penalties to the unmatched parts improves its accuracy.

Edit distance with Real Penalty (ERP) [20] combines the merits of DTW and EDR, by using a constant reference point for computing distance between gaps of two trajectories. Essentially, if the distance between two points is too large, ERP simply uses the distance value between one of those points and the reference point.

All the similarity functions discussed before are originally proposed to measure the similarity between raw trajectories, without considering the moving objects in constrained networks. However, they can be applied on the spatial trajectories on the spatial networks, by considering network distance between trajectory points rather than Euclidean distance. Indeed, several similarity measure methods have been presented aiming to study the similarity between network constrained trajectories.

Jaccard Similarity

The Jaccard similarity [73] is originally defined over the strings to compute the number of shared terms over the number of all unique terms in both strings. Jaccard similarity is not suitable for measuring the similarity between two raw trajectories. This is due to that, it may rarely happen that two coordinate points of two different trajectories be the same. While it is more suitable for trajectories that have already mapped to a network [88, 22].

Let Q, T are two trajectories on a spatial network as a sequence of nodes (or edges). The Jaccard similarity for Q and T defined as follows:

$$Sim(Q,T) = \frac{L_c(Q,T)}{L(Q) + L(T) - L_c(Q,T)}$$

Where, $L_c(Q,T)$ is the total length of the common part between trajectories Q and T, and $L(Q)$ denotes the length of the trajectory Q.

Dealing with raw trajectories, the work in [57] represents each trajectory as a sequence of cells of the grid, by using a geographical grid based on a 2D coordinate system. Then by using the Jaccard distance, they compute the

similarity between two raw trajectories.

Moreover, Won et al. [87] presented a function based on Jaccard distance in the opposite way to define dis-similarity between trajectories on the spatial network. They consider each trajectory (i.e. a sequence of segments of the road network) as a string. Given two trajectory Q, T, they propose a dis-similarity function as follows:

$$DisSim(Q,T) = \frac{L_d(Q,T)}{L(Q) + L(T)}$$

Where, $L_d(Q,T)$ is the number of disjoint segments of Q and T, and $L(Q)$ denotes the length of the trajectory Q in terms of the number of its segments.

By Jaccard similarity, we can observe that two trajectories are considered similar if they necessarily share any common part, otherwise they are dis-similar. Therefore, the main drawback of Jaccard similarity is that this measure does not take into account the proximity between trajectories, while two trajectories may be similar even if they do not share a common node.

All the aforementioned functions are basic functions to compute the spatial similarity between trajectories are constrained or un-constrained. However, the temporal aspect of a trajectory represents the movement varying with time and only considering the spatial aspect of trajectories and ignoring the temporal information is an obvious limitation in discovering the similarity of trajectories. In the next section, we focus on studies aimed at finding similar trajectories considering both spatial and temporal similarities. Most of these studies proposed the similarity functions inspired by the basic functions, cleverly.

2.3.3 Spatiotemporal Similarity

There is a general approach which is widely used in the literature to define the spatiotemporal similarity. This approach defines the different similarity functions in spatial and temporal domains and then combines both functions in a single spatiotemporal function. However, due to the different spatial and temporal distances, this approach needs to specify spatiotemporal weight parameters to achieve a combined function. Let Q, T be two trajectories and $\mathcal{D}_s(Q,T)$ and $\mathcal{D}_t(Q,T)$ denoted the spatial and temporal distances between two trajectories Q, T, respectively. The combined spatiotemporal distance $\mathcal{D}_{st}(Q,T)$ can be thereafter express as the function forms summarized in

Table 2.2. As we can see, the majority of the similarity functions are based on the spatiotemporal parameter, which must be known in advance.

Spatiotemporal function	**References**
$\sigma * \mathcal{D}_s(Q,T) + (1-\sigma) * \mathcal{D}_t(Q,T)$	[81, 79, 76, 82]
$(\mathcal{D}_s(Q,T) + \sigma * \mathcal{D}_t(Q,T))/2$	[19]
$(\mathcal{D}_s(Q,T)/\sigma + 1) * (\mathcal{D}_t(Q,T) + 1)$	[19]
$\mathcal{D}_s(Q,T) * \mathcal{D}_t(Q,T)$	[88, 74]
$\sigma * \mathcal{D}_s(Q,T) + \mathcal{D}_t(Q,T)$	[41]

Table 2.2: Spatio-temporal similarity functions: the parameter σ controls the relative importance of the spatial and temporal similarities.

Hwang et all. [41] define spatial and temporal similarity based on road networks. They predefined PoIs (Pints of Intrest) on the road network and ToIs (Time of Intrest) of moving objects. Thereafter, they define the similarity function based on the predefined PoI and ToIs. In which they assert that two trajectories are similar if they traverse the same PoI and ToI. They combine the temporal and spatial distances as shown in Table 2.2 by considering the spatial parameter as the subtraction of the speed of two trajectories.

Xia et all. [88] propose a Jaccard based similarity function by considering the spatial and temporal aspects of the trajectories in which they are constrained by the network. The definition of their similarity function is based on the length of the common parts of trajectories. Tow trajectories are considered similar if they necessarily share any common part, otherwise, they are dis-similar. The spatiotemporal similarity function in their work is defined as the spatial distance multiple the temporal distance (see Table 2.2).

In both proposed methods in [88, 41], they do not take into account the proximity of the trajectories in the similarity definition and the trajectories need to traverse the same points to consider as the similar trajectories.

Tiakas et al. [81] propose a pairwise based distance to measure the proximity of two trajectories in space as a spatial similarity. Besides, they propose a temporal similarity between two trajectories with respect to the time required to travel from one node to the next (inter-arrival times) as shown in the following example:

Example 1 *Let two trajectories T_1, T_2 with the length 3, move between the nodes of a graph as shown in Figure 2.1. The values between two nodes*

denote the time needed to traverse a node to another one for each trajectory. The temporal distance between two trajectories in the example, based on the temporal similarity function defined in [81] is as follows:

$$\begin{aligned}\mathcal{D}_t &= 1/3(|4-3|/\max\{3,4\} + |6-4|/\max\{6,4\}) + |5-5|/\max\{5,5\} \\ &= 1/3(1/4 + 2/6 + 0)\end{aligned}$$

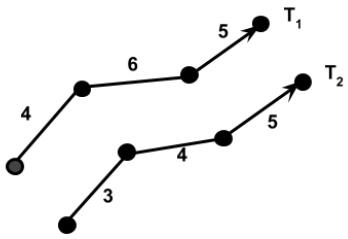


Figure 2.1: An example of the temporal distance in [81]

.

Intuitively, two trajectories are temporally similar in [81] if they move between nodes with similar speed, even if they traverse nodes at different times. Therefore, this similarity function cannot reflect the requirements for those applications that the exact time of the events is important for them. Indeed, they control the effect of these two distances with spatiotemporal parameter in a linear combination as presented in Table 2.2.

Sha et al. [74] define a spatiotemporal similarity function for trajectories on the road network. They propose the problem of finding trajectories in which they are close to a set of query locations with time stamps. For defining the similarity function, they find match pairs in trajectories for each query location and then define the pairwise based similarity function with respect to the spatial distance and time distance between only these pairs. They combine the spatial and temporal distances in a similar way to the spatiotemporal distance in [88].

Shang et al. [79] compute the similarity score between two trajectories. Each trajectory is a sequence of time-stamped nodes on the road network. They provide a distance function in the spatial and temporal domains, independently. The spatial function is an extension of LCSS as described before. They defined the temporal distance based on the LCSS. Therefore, a linear

	Temporal	Proximity	Properties	Input	Complexity time[1]
[87]	×	×	Jaccard similarity based	Two strings as trajectories	$O(\ell^2)$
[88]	✓	×	Jaccard similarity based	Two strings as trajectories	$O(\ell^2)$
[41]	✓	✓	Pair to pair distance computation only at **specific predefined points**	Two trajectories with **the same length**, implicitly	$\boldsymbol{O(\ell)}$
[81]	✓	✓	Spatial and temporal distance computation in separate way (Liner combination)	Two trajectories with **the same length**	$\boldsymbol{O(\ell)}$
[79]	✓	✓	LCSS based Liner combination of spatial and temporal distance	Two trajectories	$O(\ell^2)$
[82]	✓	✓	Linear combination of spatial and temporal distance	A set of query points and a trajectory	$O(\ell^2)$

Table 2.3: Network-based similarity measures for trajectories of ℓ nodes.

combination (see Table 2.2) of the network distance and time distance between two trajectories measures the spatial-temporal similarity between two trajectories. The similarity function is able to find the similarity between two trajectories with arbitrary length when the query trajectory treats as a sequence of weighted points.

Tiakas et al. [82] provide a spatiotemporal similarity function by combining spatial and temporal distance between a set of time-stamped query points and a trajectory on a spatial network. The spatial distance is defined as the sum of the distances of the shortest path matching pair from the trajectory to each point of the query as described in Section 2.3.1. Similar to the temporal distance in [74], the temporal distance is based on the distance between timestamps of the match nodes of the trajectory to each query node. Thereafter, they propose the spatiotemporal similarity function as a linear weighted aggregation of the spatial and temporal distance functions as shown in Table 2.2.

Indeed, the time complexity of a spatiotemporal similarity measure is an important criterion, since a similarity function could be costly when it is supposed to measure the similarity between a large set of long trajectories.

Most existing measures require quadratic time [2] for similarity computations, which shown in Table 2.3. Although, there are some studies proposing the similarity functions that require linear time to be computed with some limitations, such as Hwang et all. [41] mentioned earlier, they compute the network distance between trajectories as distances between the set of prede-

[2]We measure the time complexity of each similarity function in terms of the number of distances needed to compute by each function.

fined PoIs and ToIs; Tikas et al. [81], which compute the similarity between two trajectories with the same length. Table 2.3 summarizes the network-based spatial and spatiotemporal similarity functions and illustrated the differences between each of them.

Additionally, Frentzos et al. [32] propose spatial-temporal function DISSIM aiming at computing the similarity of time series with different sampling rates. The proposed distance is applicable on trajectories on the plane as the area of the region between two trajectories. In particular, the similarity between two trajectories is defined as the integral of the Euclidean distance between two trajectories within a time interval.

It is worth to mention that there is another kind of queries in trajectory data analytics, relying on the similarity between trajectories, called similarity join query. Given two sets of trajectories and a threshold θ, the similarity join query returns all pairs of trajectories from the two sets with similarity more than θ. For example, Shang et al. [77] by using a linear combination of spatial and temporal distances, treat the similarity join query considering the trajectories of vehicles moving in road networks. Yuan et al. [94] define trajectories as a sequence of segments on the road network and propose a spatial similarity function based on the Longest Common Road Segment (LCRS) between two trajectories to study the similarity join query.

2.4 Trajectory Data Management

For studying the trajectories and answering different kinds of queries about trajectories, it is necessary to manage trajectories in a structure to retrieve the trajectories, efficiently. There are some access methods proposed in the literature to support retrieving trajectories in an efficient way. In this section, we present an overview of indexing historical spatiotemporal data, which is the most related groups of indexing to the aim of this thesis.

R-tree Variation The most popular and classical data structure for spatial data is R-tree [37, 56]. As Figure 2.2 illustrated an example [3] , an R-tree is a height-balanced data structure. Each node of the R-tree corresponds to the Minimum Bounding Box (MBR) that bounds its children. The leaves of the tree contain pointers to the database objects instead of pointers to children nodes. R-tree is a data structure for storing spatial data, which highly supports spatial range and k-nearest neighbor (k-NN) queries. There are two strategies to traverse R-tree, depth-first and best-first [38]. The best-

[3] [56]

first strategy traverses the R-tree index from the root node and always visits the MBB with the minimum distance to the query point until it reaches the leaf node and returns it as the result. The depth-first traverses R-tree from a leaf node.

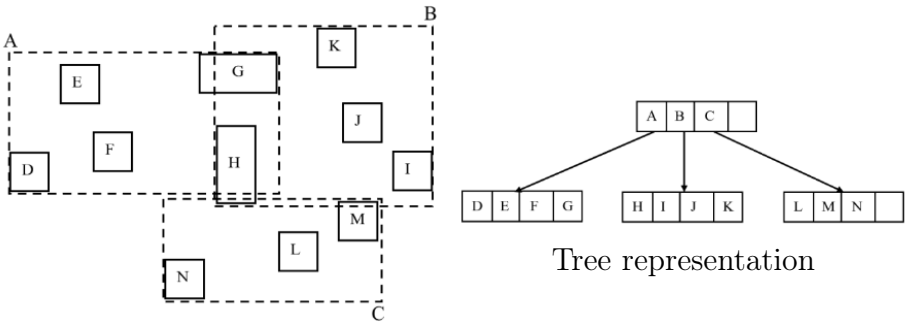


Figure 2.2: R-tree example

Although R-tree is the most popular structure that retrieves trajectories efficiently, it is not suitable for supporting spatiotemporal queries. Therefore, many studies augment the temporal aspect into R-tree to retrieve spatiotemporal trajectories efficiently. For instance, 3DR-tree [44] and STR [68] regard the time as the third dimension besides the 2D geographical space, building a 3DRtree based on trajectories. There are many other works in the literature to index the spatiotemporal trajectories which their spatial indexing structure is based on the R-tree as like SETI [17], which divides the spatial dimension into some grids, and then for each grid, they build a temporal index.

From the spatial perspective, most access methods consider that the objects are moving freely in the space. However, in several applications, the object movements are constrained (e.g., trains moving along a railroad network or vehicles moving along a road network). A few works proposed access methods for trajectories in networks, which is the most related to this thesis. We overview them as follows.

Most indexing techniques for trajectories in networks decompose the network into roads and then index the spatiotemporal location of the trajectories on each road with a specific index, which is a variant of R-tree [37]. Typical examples include FNR-tree [30] and MON-tree [24].

FNR-tree [30] uses a 2D R-tree to index road edges. For every edge in the network (i.e., every leaf entry in the 2D R-tree), there is a 1D R-tree to index the objects whose trajectories cross the edge at a certain period of time.

MON-tree [24] consists of two levels of 2D R-tree. One indexes the network edges and another one indexes the object movements along the edges. Given a spatiotemporal query, the top level R-tree is used to find the precise intersection between the spatial part of the query and the network. Based on this intersection, a set of sub-queries is generated for each intersected part of each edge involved. Then, the corresponding bottom level R-trees are accessed in order to respond to the sub-queries.

PARINET[69] PARtitioned Index for in-NEtwork Trajectories (PARINET) is an access method for retrieving the historical trajectories of moving objects over the road networks. PARINET partitions the road network and trajectory data based on the data distribution and the network topology. The time intervals for the trajectory data within each partition are indexed using a B^+-tree. They use the 2D representation of the road network. Then, based on this 2D representation they construct the topological representation of the road network. The topological representation used for partitioning the network. For 2-D range query processing, the PARINET, first finds the intersecting partitions. Then, for each accessed partition, it performs a range scan by using the B^+-tree index in order to find those trajectories that temporally overlap the given time interval.

FMI [54] FootMark Index is used for the efficient processing of the Time-Period Most Frequent Path query (TPMFP, for short). The TPMFP query identifies the most frequent path from a specified source node to a specified destination node on a network within a given time interval. This query is processed by creating a footmark graph by using the trajectories reaching the destination node within the given time interval. The footmark graph is a sub-graph of the entire network graph. The purpose of FMI is to find the trajectories according to passing the destination node within the time interval to construct the footmark graph. FMI builds a B^+-tree for each node of the graph. The B^+-tree for each node indexes the time instances the trajectories reaching to the vertex.

However, FMI provides an indexing for trajectories on graph. This structure is based on a B^+-tree for each node of the graph, which indexes the time instances of trajectories arriving in the node and they do not consider

the time intervals that a trajectory stops in a node. Similarly, the temporal indexing in PARINET[69] cannot find those trajectories that they reaching to a given specific road segment on road network out of the given time range, while having an overlap with the time interval.

TTI [101] The Time-dependent Transfer Index (TTI) is proposed to efficiently detect outliers within a given time interval. To create TTI, they divided the network into the disjoint grids. Then, they build a B-tree structure for each grid, maintaining at leaves of the tree, the trajectories passed through the grid in specified period times of the day. This structure shows each trajectory passes which grid at which time, while does not show how much a trajectory spent in a grid. Moreover, may one trajectory that stayed in a grid for a long time interval reduplicates for each entry of the tree and makes an inefficiency in the storage cost.

2.5 Trajectory-based Query Processing

Given a query object (i.e. point(s), trajectory), this kind of query aims at finding those trajectories that have the minimum distance to the query argument based on some distance functions.

Since computing the distance (similarity) between objects and trajectories is expensive, a query processing algorithm typically employs a prune-and-refine approach. The *prune* step finds a set of candidate trajectories that are likely to be the results. The *refine* step is to identify the actual query result from the set of candidates. The overall efficiency of a query processing algorithm depends on the effectiveness of the pruning step. This approach is applicable to constrained or un-constrained trajectory sets. We distinguish between the proposed methods addressing spatial and spatiotemporal queries in the Sections 2.5.1 and 2.5.2.

Another approach for efficiently addressing this kind of query is approximate searching for trajectories with minimum distance to the query object which rarely studied in the literature mentioned in Section 2.5.3.

2.5.1 Spatial Queries on Trajectories

Considering spatial trajectories, one approach to employ the prune-and-refine method on this kind of query is to treat each spatial trajectory as a set of sample points and index points of trajectories by using any spatial indexing structure (e.g. R-tree). Then, search for the sampled points in the setting of the prune-and-refine approach.

Given a point or a set of points, this approach thereafter aims at finding those sample points of trajectories either are within a distance from the query point (i.e. range query) or retrieve incrementally the nearest sample points to the query point (i.e. k-NN query). Therefore, the main task will be addressing the basic queries range and k-NN queries over a set of spatial points.

Regarding k-NN trajectories to a given point(s) query, R-tree helps to prune trajectories that are far from the query point(s) as early as possible. For instance, the works [23, 80] use R-tree to find the neighboring points of trajectories around the query location(s), by traversing R-tree by *best-first* and *depth-first* based strategies.

Chen et al. [23] propose the problem of searching trajectories by locations. The goal is to find the k-Best Connected Trajectories (k-BCT) from a database such that trajectories connect the small number of query locations in the best way. The quality of connection provided by a trajectory is measured with respect to a spatial similarity function. For query processing, they propose a pruning strategy to avoid investigating those trajectories that are far from the query locations. In their method, they index the points of all trajectories with a single R-tree [37]. They first find the closest trajectories for each query location, and then merge them for the exact k-BCT. By using the best-first and depth-first strategies of traversing R-tree, they find the k-Nearest Neighbor (k-NN) trajectories, therefore, they discover the closest trajectory to the query location. By knowing these closest points, they estimate a lower bound and upper bound of the closeness of the trajectory to the query locations w.r.t their proposed similarity function for pruning. Their proposed algorithm can efficiently find the closet trajectories to the set of the small number of query location, although it would be costly if the size of the query set is large.

Tang et al.[80] define each trajectory as a sequence of points, in which each point is mapped to a node of the road network. They aim at finding the k nearest trajectories to a given set of query points. In their proposed method, they search for each node of the query to retrieve the k-NN trajectory nodes around each query node. They use the best-first strategy and R-tree indexing for searching k-NN points of trajectories. They process each discovered trajectory in their searching process whether it is in k-nearest trajectory w.r.t the query. They evaluate the effectiveness of their method by conducting several experiments to measure the query time and I/O overhead by varying the values of k and the query size.

Moreover, [72] uses a variant of the best-first method to compute the

nearest neighbors to a query point on a spatial network. They transformed a graph representation of the network to a high dimensional space to use the spatial metrics to measure the distance between two spatial points. [70, 38] proposed depth-first and best-first based strategy, respectively, for finding k-NN point objects to a given query point. While [63] by considering a set of points as a query, proposed aggregated k-NN query.

Indeed, Papadias et al. [64] study query processing for spatial points, by using a spatial access method on the road network. It is shown that the use of Euclidean distance retrieves many candidates, and instead they propose a *network expansion* method to process range and nearest neighbor. Similarly, [82, 79] use the network expansion method for studying the spatial closeness of trajectories on the spatial network, which they described in the next section.

2.5.2 Spatiotemporal Queries on Trajectories

There are mainly two approaches to process spatiotemporal queries by employing the prune-and-refine strategy. The first is to prune trajectories in the spatial and temporal domain, separately, and then merge the discovered trajectories in a single set to process [79, 81, 41]. The second approach is to prune trajectories in the spatial domain and refine the results in the temporal domain or vise versa [41, 81, 82, 74]. However, may many trajectories be discovered from the trajectory dataset by either spatial or temporal pruning. For example, if we consider the whole lifetime of trajectories in the dataset as the time interval query, temporal pruning is not sufficient anymore. Moreover, this method is only applicable to the similarity functions that measure the spatial and temporal distances between two trajectories, independently. While, this definition of the similarity between trajectories may lead to some unnecessary distance computations, and consequently, more cost of query processing. Having one single spatiotemporal similarity function, this method could not be useful anymore. We will overview some studies based on this approach for processing spatiotemporal queries over trajectories on a spatial network (e.g. road network).

Hwang et al. [41] propose a trajectory similarity evaluation approach based on the road network distance to detect duplicates in trajectories of moving objects. The proposed pruning method is based on spatial similarity and the refining method is based on temporal distance, considering some predefined points and times of interest (i.e. PoI and ToI).

Tiakas et al. [81], for a given query trajectory on a road network, follow the two mentioned approaches to prune trajectories by each node of the query trajectory. They use M-tree to store trajectories. Based on this access method, they can efficiently retrieve trajectories are around the query trajectory in both spatial and temporal domain, since their similarity function in both domains is metric, which it is the only requirement for using the M-tree access method. In order to query processing, they first decompose the query trajectory to sub-trajectories with the same length, then for each sub-trajectory, by searching over the M-tree within the given distance radius, they find a set of candidate trajectories. Then in the first approach, they refine the candidate set by finding the close trajectories regarding the temporal distance through the searching M-tree within the given time radius. Then the compute the similarity score for each trajectory at the intersection of two achieved sets. In the second approach, they refine the candidate set by computing the spatiotemporal similarity score for each trajectory in the candidate set.

Shang et al. [79] have studied the problem of Personalized Trajectory Matching (PTM) in spatial networks. In their proposed problem, they find the most similar trajectory in the dataset to the given query trajectory. They study the problem in both spatial and temporal domains. In their algorithm, they find similar trajectories in each domain, independently, and then by integrating the two result sets, they find the trajectory with the highest similarity to the query. They present a strategy to prune the search space during query processing. They perform network expansion [26]for finding trajectories are spatially close to each node of the query. Then, by considering all timestamps of trajectory nodes on a time axis, they find those trajectories have timestamps within a range of each query timestamp. By knowing these points, they estimate lower bound and upper bound on spatial-temporal similarity to the query to reduce the search space in two domains. To avoid search space overlap between the expansion areas for each query point, they provide a strategy to choose the set of query points as the center of expansions. Thus, they perform spatiotemporal searching only for these chosen points instead of all query points. Although their method implies low computation cost by reducing the number of query points to process, the proposed method is only applicable for trajectories on the spatial networks. Indeed, if the distance between consecutive points of the query is far enough, in which their expansion areas have not any intersection, the proposed algorithm does not achieve any improvements. To accelerate the searching process, they precomputed the all pair shortest path distances.

Tiakas et al.[82] propose a prune-and-refine method, to retrieve top-k trajectories that are close to a set of query locations, spatially and temporally on a road network. They, first prune trajectories in the spatial domain, by searching on each query node. Then, they compute the similarity between the discovered trajectories and a set of query points to find the top-k most similar ones, regarding a spatiotemporal similarity function, which described earlier. In order to prune trajectories by each query node, they use a spatial index described as follows: for each node v of the road network, they store an extended adjacency list containing a set of trajectory ids traversing the outgoing edges from v. Then, by using an expansion method (Dijkstra expansion [26]), they find the neighbors of the query node, and thereafter by using the extended adjacency list assigned to the query node, find the trajectory ids are close to the query node as a candidate set. Their strategy takes only the spatial aspect of trajectories into account, to make the candidate set in the setting of the prune-and-refine method.

Sha et al. [74] study the problem of finding close trajectories to the set of query locations on the road network. In their problem, each query location is designated by timestamps. Thus, to measure the closeness of trajectories and query locations, they provide a spatiotemporal similarity function on road network as described earlier. For query processing, they prune trajectories in spatial domain, and them, they compute the spatiotemporal similarity score for only the discovered trajectories. For pruning step, they expand from each query location in the user-specified area and specify the trajectory points are in the area by using the traditional index structure (i.e. R-tree). They, therefore, investigate only those trajectories passed through these points. They take advantage of the Voronoi partitioning of the road network to precompute the distance between nodes of the road network to accelerate the query processing.

2.5.3 Approximate Query Processing

Abbasifard et al. [3] propose an approximate method to retrieve k-NN trajectories to the set of query points, by defining trajectories as a sequence of spatial sample points. Given multiple query points, they first compute the centroid of the convex hull of all query points. Then, by using the spatial index SETI, they search for k-NN sample points of trajectories to the discovered centroid regarding the distance function defined similarly to the distance function in [23] (see Section 2.3.1). In particular, they reduce the number of query points to one query point and then address the one point

query trajectory as described earlier.

2.6 PoI Prediction/Recommendation

Trajectory similarity search is an attractive and challenging topic due to its wide range of applications, such as tourism application. One of the most popular problems in this area is predicting the next movement of a tourist. The problem of predicting the most likely "*Point of Interest*" (PoI) to be visited by a tourist during its tour of a given city has been studied previously.

PoI prediction The general approach to predict the next location of users is analyzing their generated geo-tagged contents in Geo-Social Networks (GSN) like Flickr[4] and Foursquares[5] and mining trajectory patterns to devise temporally-annotated common patterns (trajectories) of movements from geo-tagged data. Trajectories are a concise representation of the behavior of moving objects as sequences of regions frequently visited with typical travel time. Trajectory-based models are exploited in [59, 8, 8], to predict the most likely locations that are of interest for a user.

In a more generic context, that of movements in a city, a similar problem is predicting the next *check-in*. Noulas et al. [61] study the problem of predicting the next venue a mobile user will visit (in foursquare-like terminology, the next *check-in*), by exploring the predictive power offered by different aspects of the user behavior. The authors propose a set of 12 features that aims to capture the factors that may drive users' movements.

Moreover, MyWay [83] predicts the future position of users using the mobility profiles, an abstract representation of the movement of users, which are based on the GPS trajectories of users and are mostly generated by vehicles. MyWay predicts the future position of a user by making use of either its mobility profile or the mobility profile of other users.

In the context of touristic trajectories, in [60] authors propose a broader set of features, originated from a Flickr dataset, capturing more dimensions of the touristic behavior. They cast the prediction problem into a "*learning to rank*" task, which allows us to use two effective Machine Learning techniques (Ranking SVM [43] and GBRT [100]) to solve it.

PoI recommendation Similar efforts have been spent on solving the PoI recommendation task. Here, the problem deals with generating a list of possible PoIs that are of interest to a tourist. It differs from the prediction task

[4] `https://www.flickr.com/`

[5] `https://foursquare.com/`

as it aims at maximizing the satisfaction of the user during its tour of the city, while the first one aims at identifying only one PoI as the first candidate to be visited. In [47], a location-aware recommender system (LARS) that uses location-based ratings to produce recommendations is proposed. Ye et al. [91] realize location recommendation services for large-scale location-based social networks, by exploiting the social and geographical characteristics of users and locations/places in friend-based collaborative filtering (FCF) approach. Ying et al. [93] propose a time-aware metric embedding a method to recommend the next point of interest to the user based on user history check-in data. In [90] a social and sequence-aware next POI recommendation model is proposed that uses the social information and the geographical location information of users. Zheng et al. [99] perform travel recommendations by mining multiple users' GPS traces. They model multiple users' location histories with a tree-based hierarchical graph on which they use a HITS-based inference model. Lucchese et al. [52] propose an algorithm that interactively generates personalized recommendations of touristic places based on the knowledge mined from photo albums and Wikipedia. The authors introduce the model as a graph-based representation of the knowledge, and exploits random walks with a restart to select the most relevant PoIs for a specific user.

Several other works tackle sub-problems of the next PoI prediction problem such as: predicting the next location in unfamiliar places by focusing on contextual factors such as weather, transportation means, place of residence, and time [55], predicting the next location from irregular patterns exploiting association rule mining [12], exploiting visual contents for PoI recommendation [86], or even predicting the next PoI category rather than the actual PoI by using NLP models [95].

Newer methods exploit deep learning algorithms in order to predict the next PoI, [89, 96, 5], usually based on recurrent and/or sequential models. However, for these methods to be successful a lot of data samples are required. In the context of next PoI prediction for tourism, the number of examples varies from city to city and even so often the examples are in the order of thousands or tens of thousands, not enough for building robust neural models.

Chapter 3

Preliminaries and Definitions

In this chapter, we present the basic concepts and definitions that are necessary to develop the techniques in this thesis. We explain the main problem that this thesis focuses on it. Finally, we propose an algorithm as a baseline method for solving the proposed problem. This algorithm will be used in the remaining of this thesis as a baseline for benchmarking our solutions.

3.1 Preliminaries

In this section, we present some basic concepts and definitions that are necessary to develop the algorithmic techniques. The most frequently used symbols are summarized in Table 3.1.

3.1.1 Trajectory Definition

A trajectory is a sequence of network nodes. These nodes usually represent points of interest (POIs), specific locations that have a special meaning or any other network locations that should be registered as *visited* by a user. Let $\mathcal{T}$ be a set of trajectories in a network, which is represented by a connected and undirected graph $G(V, E)$, where V is the set of vertices and E is the set of edges. Formally, each trajectory $T \in \mathcal{T}$ is defined as follows:

Definition 1 (Trajectory) *Given a graph $G(V, E)$, a trajectory T is defined as a sequence of pairs of the form (v_i, t_i), where v_i is a node and t_i a time interval, i.e., $T = \langle (v_1, t_1), (v_2, t_2), \cdots, (v_l, t_l) \rangle$ such that, $1 \leq i \leq l-1$, we have $(v_i, v_{i+1}) \in E$ and t_i and t_{i+1} are two consecutive time intervals. We*

Symbol	Interpretation
G	a graph
V	set of graph nodes
n	number of graph nodes ($n = \|V\|$)
E	set of graph edges
m	number of graph edges ($m = \|E\|$)
D_G	the diameter of G
$\mathcal{T}$	set of trajectories
T_i	the i-th trajectory ($T_i \in \mathcal{T}$)
$t_i = [s_i, e_i]$	the i-th time interval ($s_i \leq e_i$)
Q	the query trajectory
k	number of results
t	a time interval
$dist(v, T_i, t)$	distance between node v and trajectory T within t
$Sim(T_i, T_j, t)$	similarity between two trajectories T_i, T_j within t
$d(u, v)$	shortest path distance between nodes u and v
$\|T\|$	length of trajectory T
$T[t]$	trajectory T within time interval t
$T(i)$	trajectory T at time instance i

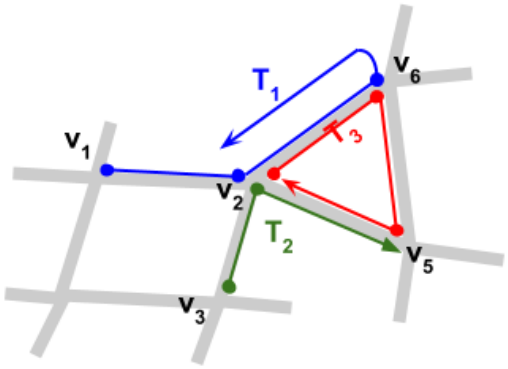
Table 3.1: Frequently used symbols.

call $|T| = l$ the description length of T, which corresponds to the number of (non-distinct) nodes traversed by T. Letting $t_1 = [s_1, e_1]$ and $t_l = [s_l, e_l]$, we refer to s_1 and e_l as the starting time and ending time of T.

We say that two intervals $t_i = [s_i, e_i]$ and $t_{i+1} = [s_{i+1}, e_{i+1}]$, with positive integer endpoints, are *consecutive* if $s_i \leq e_i < s_{i+1} \leq e_{i+1}$ and $e_i + 1 = s_{i+1}$. Given a trajectory $T \in \mathcal{T}$, we denote by $t_i = [s_i, e_i]$ the i-th time interval of T. Also, let $s = s_1$ and $e = e_l$ be the starting and ending time of T. Given a time instance $i \in [s, e]$, the notation $T(i)$ indicates the unique node $u \in V$ such that there exists a pair $(u, t) \in T$ with $i \in t$.

Example 2 *Figure 3.1 shows an example of the set of trajectories $\mathcal{T}$ containing three trajectories T_1, T_2, T_3 moving through a graph with a set of nodes $V = \{v_1, v_2, v_3, v_4, v_5\}$.*

Given a query trajectory Q, a set of trajectories $\mathcal{T}$ and an integer k, we are interested in detecting the k trajectories from $\mathcal{T}$ that have the highest



$$\mathcal{T} = \{T_1, T_2, T_3\}$$
$$T_1 = \langle (v_1, [2, 4]), (v_2, [5, 9]), (v_4, [10, 12]), (v_2, [13, 15]) \rangle$$
$$T_2 = \langle (v_3, [1, 6]), (v_2, [7, 11]), (v_5, [12, 16]) \rangle$$
$$T_3 = \langle (v_2, [1, 3]), (v_4, [4, 7]), (v_5, [8, 13]), (v_2, [14, 16]) \rangle$$

Figure 3.1: An example of a trajectory set containing three trajectories T_1, T_2, T_3 over the graph $G(V, E)$ with $V = \{v_1, v_2, v_3, v_4, v_5\}$

.

degree of similarity with respect to Q. First, we define the distance between a node v and a trajectory $T \in \mathcal{T}$. We further establish a function to rank the trajectories that are close to the given one by defining an appropriate similarity measure.

3.1.2 Trajectory Similarity Measure

This section is devoted to introducing our similarity function. It is arguably natural to assess that two trajectories are *similar* if they pass close to each other in close moments, without enforcing the too restrictive requirement of sharing common nodes or having the same length. The motion of trajectories in this thesis is restricted by the network. Thus, the similarity function must use the graph to measure the proximity between trajectories. As the Euclidean distance is not appropriate to measure the distance of nodes on the graph, it is important to use the graph distance metric instead. Moreover, by taking into account only the visited nodes of trajectories over a graph, two trajectories are determined as similar while they are not similar considering time aspect. Therefore, the similarity measure must consider both aspects of the trajectories: temporal aspect and the location of trajectories over the

graph, i.e the *structural* aspect.

In summary, a similarity measure for trajectories on a graph has the following requirements:

1. The similarity function must be able to compare two trajectories with different lengths (i.e. different sampling rates).

2. Both temporal and structural aspects must be taken into account by the similarity function.

3. The similarity function must measure the proximity between two trajectories (within the time intervals).

In order to define the building block of our similarity measure, we need first to restrict trajectories within a time interval, as shown in the following definition.

Definition 2 (Time Restricted Trajectory) *Given a trajectory T and a time interval $t = [s, e]$, the time restricted trajectory $T[t]$ is the sequence of pairs $(u_i, t_i) \in T$ such that $t_i = [s_i, e_i]$ has overlap with $t = [s, e]$ (i.e. $t_i \cap t \neq 0$).*

Without loss of generality, we assume that $\sum_{(u_i,t_i)\in T[t]} |t_i| = |t|$, where, $|t| = e - s$ and $|t_i| = e_i - s_i$.

We define the distance between a *node* v and a *trajectory* T within a *time interval* t as follows:

$$dist(v, T, t) = \frac{\min_{(u_i,t_i)\in T[t]} d(v, u_i)}{D_G} \tag{3.1}$$

where D_G is the diameter of the graph G (i.e. the maximum shortest path distance from any possible pair of nodes in the graph).

Proposition 1 *The value $dist(v, T, t)$ is always in the interval $[0, 1]$.*

We observe that the extreme values are achieved in the following cases.

Property 1 $dist(v, T, t) = 0$ if and only if there exists at least one time instance $i \in t$ such that $T(i) = v$.

Property 2 $dist(v, T, t) = 1$ if and only if for each time instance $i \in t$, node $T(i)$ is at distance D_G from v. This corresponds to the case where T spends the whole time interval t on nodes of G that are at maximum distance from v.

We are now ready to introduce our similarity measure, using the distance function defined in Equation 3.1.

Taking inspiration from [23], we aim at assigning a larger contribution to those parts of the trajectories that are close for sufficiently long time intervals (while assigning lower contribution to farther parts). These desired properties are satisfied as follows.

Definition 3 (Similarity Function) *Given a query trajectory Q and a target trajectory $T \in \mathcal{T}$ and a time interval t, the similarity of T with respect to Q within t is*

$$Sim(Q, T, t) = \frac{\sum_{(v_i,t_i)\in Q[t]} |t_i| \times e^{-dist(v_i,T,t_i)}}{|t|} \tag{3.2}$$

Lemma 1 *The similarity function $Sim(Q, T, t)$ is always in the interval $(0, 1]$.*

Proof 1 By Proposition 1 for each $(v_i, t_i) \in Q[t]$ we have:

$$0 \leq dist(v_i, T, t_i) \leq 1 \rightarrow 1 \geq e^{-dist(v_i,T,t_i)} \geq e^{-1} > 0$$
$$\xrightarrow{\times|t_i|} 0 < |t_i| \times e^{-dist(v_i,T,t_i)} \leq |t_i|$$

By summation over each pair $(v_i, t_i) \in Q[t]$ we get:

$$0 < \sum_{(v_i,t_i)\in Q[t]} |t_i| \times e^{-dist_t(v_i,T,t_i)} \leq \sum_{(v_i,t_i)\in Q[t]} |t_i| \tag{3.3}$$

Assume that $\sum_{(v_i,t_i)\in Q[t]} |t_i| = |t|$: dividing the equation 3.3 by $|t|$ we get:

$$0 < \frac{\sum_{(v_i,t_i)\in Q[t]} |t_i| \times e^{-dist(v_i,T,t_i)}}{|t|} \leq 1$$

■

For two trajectories Q, T and time interval t, we have:

Property 1 If $Q[t] = T[t]$ then $Sim(Q, T, t) = 1$.

Property 2 $Q[t] = T[t]$ iff for each $i \in t$, $Q(i) = T(i)$

Lemma 2 *Given two trajectories Q and T, and a time interval t, where $|Q[t]| = \ell_1$ and $|T[t]| = \ell_2$, computing $Sim(Q, T, t)$ requires $O(\ell_1 + \ell_2)$ time and pairwise node distances.*

Proof 2 Looking at equations (3.1) and (3.2), it seems that $O(\ell_1 \times \ell_2)$ computation is needed. The cost is instead $O(\ell_1 + \ell_2)$ if we realize that the computation is conceptually a nested loop in which the nodes in Q and T are scanned forward when a pairwise distance $d(v_i, v)$ is needed: in each iteration at least one node is scanned, thus the total cost is $O(\ell_1 + \ell_2)$.

3.1.3 Top-k Most Similar Trajectories

Given a set of trajectories $\mathcal{T}$, we define the problem of retrieving the top-k similar trajectories to a given one in a specific time interval. More formally, this desired set of trajectories, referred to as K-MsTraj, corresponds to the following one.

Definition 4 (k-Most Similar Trajectory(k-MSTraj)) *Given a set of trajectories $\mathcal{T}$, a query trajectory Q and a query time interval t, the set $\mathcal{T}' \subset \mathcal{T}$ with $|\mathcal{T}'| = k$ is the k-MSTraj if for each trajectory $S \in \mathcal{T}'$ the following assumption holds:*

$$\forall T \in \mathcal{T} - \mathcal{T}' : Sim(Q, S, t) \geq Sim(Q, T, t)$$

To present a good intuition of our proposed similarity function, we provide a representative example of four trajectories chosen randomly in a dataset of trajectories moving in Milan (See Chapter 4 for more details about this dataset). By using the similarity function in Definition 3, for the red trajectory in Figure 3.2 as the query trajectory, the green trajectory has the maximum similarity among other trajectories in Figure 3.2, i.e. hence it is a solution for the K-MsTraj problem with $k = 1$. Note that the trajectories in red, green, yellow and violet in Figure 3.2, start to move at time instances (in msec) 37237, 45964, 57354 and 26430, and stop the movement at 582313, 331565, 57872 and 564740, respectively.

Figure 3.2: An example including 4 random trajectories in a dataset of trajectories moving in Milan. The trajectory with the red color is a query. The green one is the most similar one by Definition 3.

A straightforward approach to find K-MsTraj is to compute the similarity score for each trajectory $T \in \mathcal{T}$ and Q, reporting the k trajectories

with maximum scores. Clearly, we only consider those trajectories that are defined for all instants $i \in t$. This approach is inefficient, as it requires the computation of $O(|\mathcal{T}| \times \max\{|T|_{max}, |Q|\})$ shortest path distances, where $|\mathcal{T}|$ and $|T|_{max}$ are respectively the number of trajectories and the maximum length of trajectories in $\mathcal{T}$.

In this thesis, we aim at accelerating this process by managing a data structure to retrieve the trajectories, structurally and temporally, in an efficient way. To this end, we provide an indexing structure described in Section 3.2, which is a basic indexing structure used by the proposed strategies in this thesis.

3.2 NTrajI Indexing

The *Neighborhood Trajectory Indexing* (NTrajI) described here efficiently finds the closest trajectories with respect to each node of the query and its corresponding time interval.

Interval Tree

For building NTrajI we use an interval tree, which is a binary tree to store a set of intervals based on the median of the endpoints of the intervals. In this structure, all the intervals intersect the median point are stored at the root of the tree. The intervals lying completely to the left and right of the median point, are respectively stored in the left subtree and the right subtree of the root. The subtrees are constructed recursively in the same way. By using this structure we are able to find efficiently all intervals that overlap with any given interval or point using the following well-known result.

Theorem 1 ([11]) *Given a set of n intervals, an interval tree uses $O(n)$ space. It can be built in $O(n \cdot \log n)$ time and can report all intervals that overlap a query interval or point in $O(\log n + k)$ time, where k is the number of reported intervals.*

In NTrajI, we build an Interval Tree IT_v for each node $v \in V$ to store the time intervals that each trajectory in dataset spent in v and its neighbors, and maintain the corresponding trajectory ids. Specifically, we define this set of time intervals as a projection set of each node as follows.

Definition 5 (Node Projection Set) *The projection set S_v of a node v is the set of pairs (t, T) of all trajectories $T \in \mathcal{T}$ that pass through the nodes $\{v\} \cup N(v)$, during the time interval t. Namely, $S_v = \{(t, T) \mid (u, t) \in T$ and $u \in \{v\} \cup N(v)$ and $T \in \mathcal{T}\}$.*

The Interval Tree IT_v maintains all pairs $(t, T) \in S_v$ for each node $v \in V$. Each entry of IT_v is of the form $\langle t, id \rangle$, where id is the trajectory identifier, and t is the time interval the trajectory id spent in $\{v\} \cup N(v)$. Note that, there can be more than one pair associated with node v with the same trajectory id since each trajectory can traverse a vertex multiple times.

Example 3 *Considering the trajectory set in Example 2, we show the* NTraj *in Figure 3.3, storing the trajectories passing through the node v_1 in Example 2. For the node $v_1 \in V$, the IT_{v_1} maintains all pairs in the projection set $S_{v_1} = \{(T_3, [1, 3]), (T_1, [2, 4]), (T_1, [5, 9]), (T_2, [7, 11]), (T_1, [13, 15]), (T_3, [14, 16])\}$.*

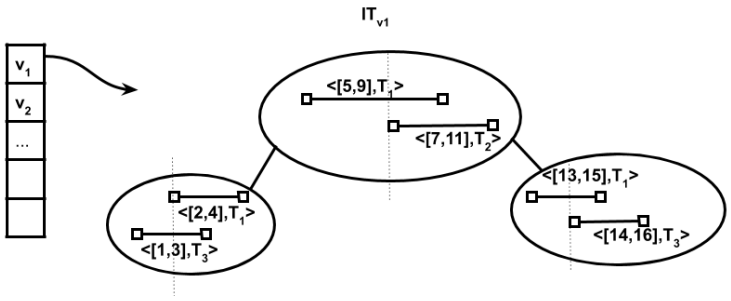


Figure 3.3: An example of NTrajI

.

By Theorem 1, we can derive that NTrajI uses $O(|\mathcal{T}| \times |T|_{max} \times \Delta)$ space, where Δ denote the maximum degree of G. For a given node v and a time interval t, let $\Gamma_{(v,t)}$ denote the trajectories that traverse either v or $N(v)$ within t. By searching over the NTrajI, we are able to find $\Gamma_{(v,t)}$ efficiently by taking $O(\log |\Gamma_v| + |\Gamma_{(v,t)}|)$ time, where $|\Gamma_v|$ is the size of IT_v and $|\Gamma_{(v,t)}|$ is the number of reported trajectories.

3.3 Baseline: Exact Computation of k-MsTraj

In this section, we introduce a baseline method to solve exactly the k-MsTraj problem. This is based on an indexing phase, described in Section 3.2, which aims at accelerating the query processing. We will use this method as a baseline in the experimental evaluation of our proposed approaches in this thesis.

BaseLine Method

We introduce a pruning technique as the BaseLine method to k-MsTraj problem (see Algorithm 1). The BaseLine method, explores the set Γ of trajectories that are most promising to be k-MsTraj. They are discovered by searching the NTrajI index. Then, by computing the similarity score of each trajectory in Γ, this method finds the trajectories having the highest similarity with Q within the time interval t.

Therefore, the main task is to construct the *candidate set* Γ using NTrajI. In particular, for a given query trajectory Q, for each $(v_i, t_i) \in Q$, we aim at finding the trajectories that are close to v_i within t_i. To this aim, for each $(v_i, t_i) \in Q$, we search for $\Gamma_{(v_i,t_i)}$ in IT_{v_i}. The union of all discovered trajectories makes the candidate set Γ for k-MsTraj w.r.t Q.

As illustrated in Algorithm 1, we restrict query Q within time query t. Then we search over IT_{v_i}, for each $(v_i, t_i) \in Q[t]$, by NTrajI-search(v_i, t_i) to build $\Gamma_{(v_i,t_i)}$. So we have $\Gamma = \bigcup_{(v_i,t_i)\in Q[t]} \Gamma_{(v_i,t_i)}$.

Proposition 2 *By construction,* k-MsTraj $\subseteq \Gamma \subseteq \mathcal{T}$.

We compute the similarity score, with respect to the function in Definition 3, for each trajectory in Γ. In order to maintain the k trajectory ids with the highest similarity score during the search process, we use the heap H of size k, which contains the k-MsTraj at the end of the searching process.

Lemma 3 *Given a set of trajectory $\mathcal{T}$, a query trajectory Q and a query time interval t, the* BaseLine *method solves the* k-MsTraj *query in $O(C_\Gamma + |\Gamma| \times \max\{|T|_{max}, |Q[t]|\})$ time, where $|T|_{max}$ is the maximum length of trajectories in Γ, and C_Γ is the time cost of computing Γ, that is, $O(\sum_{(v_i,t_i)\in Q[t]} \log |\Gamma_{v_i}| + \log |\Gamma_{(v_i,t_i)}|)$.*

Algorithm 1: BASE

Input: Graph G, set of trajectories $\mathcal{T}$, query trajectory Q, time interval $t = [a, b]$, integer k

Result: k-MsTraj

H /* H is a heap in size of k */
$Q[t]$ /* restricted Q within t */
$\Gamma_{(v_i,t_i)} \leftarrow$ NTrajI-search(v_i, t_i) for each $(v_i, t_i) \in Q[t]$
$\Gamma = \bigcup_{(v_i,t_i)\in Q[t]} \Gamma_{(v_i,t_i)}$
while $heap - size(H) < k$ **do**
    H.add(T), for $T \in \Gamma$
for $T \in \Gamma - H$ **do**
    if $Sim(Q, min(H), t) < Sim(Q, T, t)$ **then**
        H.remove(min(H)) /*min(H) returns a trajectory id in H with minimum score */
        H.add(T)
k-MsTraj $\leftarrow$ trajectories in H

Chapter 4

Trajectory Similarity Search: Shrinking Approach

In this chapter, we propose an approximate method for answering k-MsTraj query, that is, identifying the trajectories are most similar to a given query trajectory within a given time interval. Naively, to answer such queries, it is needed to compute the similarity score for all trajectories in the dataset. This would be costly for a large volume of data since a typical trajectory similarity query needs to compute a large number of similarity scores, where each one might compute hundreds or thousands of shortest path distances. Since computing the network distance (i.e. shortest path distance) between trajectories is expensive, a query processing algorithm typically aims at minimizing the number of distance computations. In this study, we aim at proposing an approximate method to compute the similarity between trajectories by representing trajectories with a fewer number of nodes and then consequently, reducing the number of shortest path distance computations with the minimum number of precomputation distances.

In the proposed method, firstly, we divide the network into a set of Voronoi groups [28]. Then, we shrink each trajectory considering centers of the Voronoi groups. In the real-world application, it is applicable to represent trajectories by centers as, Points of interest (PoI) in a city, the most crowded nodes on a road network, and the nodes with the most traffic on the Internet networks. For query processing, we retrieve the nearby trajectories around the Voronoi centers by using an extension of the indexing structure proposed in Section 3.2. After that, we estimate the similarity between retrieved trajectories by using the Voronoi centers. Moreover, to accelerate the query processing, we take benefit of the Voronoi partitioning to recompute the shortest distance between centers.

In compared to the state-of-the-art [78, 79] that precomputes the all-to-all pairwise node shortest path distances (see Chapter 2), we only compute a linear number of shortest path distances in the precomputation phase. We show in Section 4.2 that this cost of precomputing is negligible.

The chapter is organized as follows. In Section 4.1 we explain the approximate method with more details. Thereafter, in Section 4.2, we evaluate the effectiveness of the proposed algorithm in terms of the query time and accuracy of the proposed methods. Due to the lack of competitors, as there are no linear-time similarities on trajectories on a graph that uses flexible proximity taking into account the temporal aspect of trajectories, we evaluate the performance of the methods in comparison with the BaseLine method proposed in Section 3.3.

4.1 Approximate Computation of k-MsTraj

We propose the approximated methods with two-phase preprocessing in Section 4.1.1.

- First, we partition the underlying graph G into disjoint groups of nodes, precomputing distances from the centers of each group. As the similarity function in Definition 3 uses the shortest path distance between nodes of the graph, we aim at approximating distances between trajectories using the distances between the centers of the groups.

- Second, we adapt the NTrajI indexing, described in Section 3.2, so that we maintain trajectories among the groups in a structure called VoTrajI.

For the query processing, given a query trajectory and a given time interval, we show how to estimate the similarity scores for the trajectories in $\mathcal{T}$ using the partitioning and the new VoTrajI index in Section 4.1.2

4.1.1 Two-phase Preprocessing

The partitioning takes into account the popularity of the nodes, by choosing the centers of the groups as the nodes having a higher number of trajectories passing through them. To do this, it uses Voronoi Diagrams for graphs (VDG), as explained next.

The VDG is a generalization of the classic Voronoi diagram. For graph $G = (V, E)$ and a set of trajectory $\mathcal{T}$, let $\mathcal{C} = \{c_1, c_2, \dots c_h\}$ be a set of h (most frequent) nodes in V, called Voronoi sites i.e. center nodes. The VDG

over the nodes in $\mathcal{C}$ is defined as a partition of V into h groups $g_1, g_2, \cdots, g_h$, one for each center in $\mathcal{C}$. Node $u \in V$ is in group g_i with center c_i (i.e. $g_i.\mathcal{C} = c_i$) iff $d(u, c_i) \leq d(u, c_j)$ for each $c_j \in \mathcal{C}$ with $i \neq j$ (ties are broken arbitrarily). We can divide G into h Voronoi groups in $O(n \log n)$ time when $h = O(n^\epsilon)$ for a positive constant $\epsilon < 1$ [28].

Once the Voronoi groups have been computed, we precompute and store the pairwise distances among the center nodes of these groups. We aim at using these distances as an approximation for the distances required by the similarity function in Definition 3. By running one BFS for each center node, we compute the distance between each pair $c_i, c_j \in \mathcal{C}$ in $O(m \cdot n^{1/2})$ time, where we set $h = n^{1/2}$. As a result, we obtain the following lemma.

Lemma 4 *Graph partitioning and centers' distance precomputation require $O(m \cdot n^{1/2})$ time. The space required by the centers' distance table is $O(n)$ space.*

We now discuss how to provide the *Voronoi Trajectory Indexing* (VoTrajI) by adapting the NTrajI data structure. We build an interval tree IT_c for each $c \in \mathcal{C}$. Interval tree IT_c stores the time intervals of trajectories in $\mathcal{T}$ spent within the nodes in g, when $g.\mathcal{C} = c$. The VoTrajI maintains the corresponding trajectory ids of the time intervals. By expanding the definition 5, we have:

Definition 6 (Group Projection Set) *The projection set S_c of a center node $c \in \mathcal{C}$ stores the pairs (t, T) of all trajectories $T \in \mathcal{T}$ that pass through the nodes in g, when $g.\mathcal{C} = c$, namely, $S_c = \{(t, T) \mid (v, t) \in T$ and $v \in g$ and $g.\mathcal{C} = c$ and $T \in \mathcal{T}\}$.*

The Interval Tree IT_c for each node $c \in \mathcal{C}$ maintains all pairs $(t, T) \in S_c$. Each entry of IT_c is the form of $\langle t, id \rangle$, where id is the trajectory id, and t is the time interval that the trajectory id spent at $v \in g$, where $g.\mathcal{C} = c$.

To reduce the storage space used by IT_c, for each $c \in \mathcal{C}$, we consider a sequence of consecutive time intervals with the same trajectory id in S_c as a single time interval with the corresponding trajectory id.

Lemma 5 *The two-phase preprocessing takes $O(m \cdot n^{1/2})$ time and $O(n)$ space.*

4.1.2 Query Processing

Considering how a trajectory $T \in \mathcal{T}$ is represented with respect to the center nodes of the Voronoi diagram, let $(v_i, t_i) \in T$ and $v_i \in g$, where g is a Voronoi

group of G. We represent $(v_i, t_i) \in T$ as (c, t_i) where $g.\mathcal{C} = c$. We obtain a new trajectory T' as a sequence of center nodes and the corresponding time intervals. Note that T' can traverse a sequence of the nodes belong to the same Voronoi group within consecutive time intervals. To avoid the duplication of nodes for consecutive time intervals, we define *shrunk trajectories*.

Given a trajectory $T' = \langle (c_1, t_1), \ldots, (c_l, t_l) \rangle$, consider the operator $\textsc{shrink}(T')$, which recursively merges any pair $(c_i, t_i), (c_{i+1}, t_{i+1}) \in T'$ as $(c_i, t_i + t_{i+1})$ when $c_i = c_{i+1}$ and t_i, t_{i+1} are two consecutive time intervals (here operation $t_i + t_{i+1}$ gives $[s_i, e_{i+1}]$).

Definition 7 (Shrunk Trajectory) *Let $T = \langle (v_1, t_1), \ldots, (v_l, t_l) \rangle$ be a trajectory in $\mathcal{T}$. Consider the corresponding sequence $T' = \langle (c_1, t_1), \ldots, (c_l, t_l) \rangle$ with respect to the Voronoi groups. We define the shrunk trajectory of T as $\hat{T} = \textsc{shrink}(T')$.*

Note that it takes $O(l)$ time to obtain $\hat{T}$, and that $|\hat{T}| \leq |T|$.

We consider two variants for estimating k-MSTraj.

1. Shrunk query (SHQ): Shrinking trajectory Q during query time.
2. Shrunk query and target (SHQT): Shrinking each trajectory in $\mathcal{T}$ during the preprocessing and shrinking trajectory Q during query time.

Both variants perform a search on the VoTrajI index using the shrunk query trajectory $\hat{Q}$. The outcome of that search is a set $\tilde{\Gamma}$, which is defined as Γ in Section 3.3, except that we use VoTrajI in place of NTrajI. This makes a difference, as the property in Proposition 2 does not necessarily hold anymore. Indeed there could be a trajectory $T \in$ k-MsTraj such that $T \notin \tilde{\Gamma}$ (whereas surely $T \in \Gamma$). This approximated version has the advantage of speed, which motivates this study.

Variant SHQ

In this variant, we compute the similarity scores for each trajectory $T \in \tilde{\Gamma}$ w.r.t the shrunk query $\hat{Q}$. In particular, we make an estimate of $Sim(Q, T, t)$ as $Sim(\hat{Q}, T, t)$, and report the top-k trajectories with the highest estimated similarity score. To measure the precision ratio of this estimation, the similarity function makes an estimate of $d = d(v, u)$ as $\hat{d} = d(c, u)$, when $v \in Q$, $u \in T$ and c is the center node of a group that includes v.

Lemma 6 *Let $v, u \in V$. The ratio between $d = dist(v, u)$ and $\bar{d} = dist(c, u)$, where c is the center of Voronoi group containing v, is bounded as $1 \leq d/\bar{d} \leq 3$.*

Proof 3 Let $dist(c, v) = r$. We have two possibilities. If $r \leq 2\bar{d}$, then by triangle inequality $d \leq r + \bar{d}$ and thus $d \leq 3\bar{d}$. Else if $2\bar{d} < r$, then by triangle inequality $r \leq d + \bar{d}$ and thus $\bar{d} < d$. ■

Although we reduce the number of nodes in query trajectory which needed to be processed, the number of distance computations is still large. As mentioned earlier, the cost of distance computation depends on the length of the trajectories within the query time interval t. In order to reduce this cost, we consider our second variant SHQT.

Variant SHQT

In this variant, we estimate $Sim(Q, T, t)$ as $Sim(\hat{Q}, \hat{T}, t)$. Specifically, the similarity function makes an estimate of $d = d(v, u)$ as $\tilde{d} = d(c_i, c_j)$, when $v \in Q$, $u \in T$ and c_i, c_j are the center nodes of the groups that include $v \in Q$ and $u \in T$, respectively.

Lemma 7 *Given two nodes v, u belonging to Voronoi groups with center nodes c_i, c_j, respectively, the ratio between $\tilde{d} = dist(c_i, c_j)$ and $\bar{d} = dist(c_i, u)$ is bounded as $\tilde{d}/\bar{d} \leq 2$.*

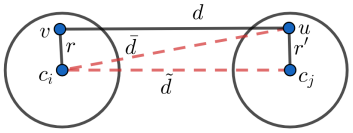


Figure 4.1: $c_i \in g_i$ and $c_j \in g_j$ such that $c_i, c_j \in \mathcal{C}$

Proof 4 As illustrated in Figure 4.1, let $dist(v, c_i) = r$ and $dist(u, c_j) = r'$. We consider the groups g_i, g_j containing the two centers c_i, c_j, respectively. By triangle inequality we have $\tilde{d} \leq r' + \bar{d}$. Since $u \in g_j$ and $u \notin g_i$ then $r' \leq \bar{d}$. Thus, $\tilde{d} \leq 2\bar{d}$. ■

Using Lemma 6 and 7, we are able to conclude that $\tilde{d} \leq 2d$ when $r > 2\bar{d}$. If $r > 2\bar{d}$ by triangle inequality we have $\bar{d} < d$. Since $\tilde{d} \leq 2\bar{d}$, we can obtain that $\tilde{d} \leq 2d$.

The similarity function in Definition 3 assigns a larger contribution to those nodes of the trajectories that are closer rather than the farther ones. Thus, by Lemma 6 and 7, we expect that the estimated similarity score in both variants would be larger than the exact similarity score. We will evaluate this idea in our experiments.

4.2 Experimental Evaluation

This section is devoted to comparing the performances of SHQ and SHQT with respect to the Baseline method, hereafter called the BASE. The evaluation aims at the following questions.

Q1: How fast is getting the answer for a query, i.e. how much is the query time?

Q2: How fast is the preprocessing time?

Q3: How good is the quality of the solution found if compared with the exact solution?

Each one is discussed in Section 4.2.2, 4.2.3, and 4.2.4, respectively.

We will answer these questions by evaluating the performance of each method by varying the value of k. In particular, we set k as 2^i for $i = 0, \ldots, 6$. Each experiment requires a graph, a trajectory set, a query trajectory, and a query time interval. For each experiment, we choose 100 trajectories as query trajectories, randomly. For each query trajectory, we set the query time interval as the whole lifetime of the query to study the wide range of queries.

Our computing platform is a machine with Intel(R) Xeon(R) CPU E5-2620 v3 at 2.40GHz, 24 virtual cores, 128 Gb RAM, running Ubuntu Linux version 4.4.0-22-generic. The program has been written in `Python3`.

4.2.1 Datasets

We conduct our experiments on two real trajectory datasets, and one synthetic dataset over a real-world graph, described as follows. The main properties of each dataset are shown in Table 4.1.

Synthetic Trajectory Set: Facebook and WEB network We manage our experiments over synthetic trajectories on a real Facebook and WEB network. The main properties of the networks are shown in the Table 4.1 and in Figures 4.2(a) and 4.3(a), where we show the degree distribution of the networks. We generated a trajectory dataset $\mathcal{T}$ composed by 1000 and 100k trajectories, each one traversing the nodes of Facebook and WEB networks ,respectively, using the trajectory generator shown in Algorithm 2, where the maximum duration (MaxDuration) is set to 30 and the maximum life

Algorithm 2: TRAJECTORY GENERATOR

Input: $G(V, E)$
Result: $\mathcal{T}$

```
N /* The number of trajectories */
MaxTime /* The maximum life time of trajectories */
MaxDuration /* The maximum time a trajectory can stays at each
  node */
t=0 /* The time that a trajectory starts to move */
for each trajectory id in range(1,N) do
    vrn ← choose randomly a node in V
    assign (vrn, t) to trajectory id
    t ← t + random number in range(1, MaxDuration+1)
    while t < MaxTime+1 do
        vrn ← choose randomly a node in Neighbor(vrn)
        assign (vrn, t) to trajectory id
        t ← t + random number in range(1, MaxDuration+1)
```

(MaxTime), i.e. the number of timestamps a trajectory is active, is set to 4000.

The average length of the generated trajectories in terms of the number of nodes in Facebook and WEB networks are about 260 and 194, respectively. Moreover, Figures 4.2(b) and 4.3(b) show the popularity of each node of the graph, in terms of the number of trajectories traversing through each node in both Facebook and WEB networks. Similarly, Figures 4.2(c) and and 4.3(c) show the distribution of the length of the trajectories generated, which seems to follow a Gaussian distribution.

Real Trajectory Set: Milan We conduct our experiments on a dataset based on tracks of private cars in Milan. The dataset contains about 165.000 trips of about 17.000 users in the temporal windows 2-8 April 2007. Globally, the dataset consists of about 2 million GPS observations, each consisting of $\langle userid, datetime, lat, lon \rangle$, where *userid* is the car identifier, *datetime* is the time of the observation and lan, lon, is the spatial coordinate. The sequence of time-ordered with the same user *id* makes the *raw trajectory* for the user *id*. Then, the raw trajectories need to be mapped to a network. To this end, we use the k-means clustering technique for GPS points clustering. Then, we use the k-means clusters to represent each raw trajectories w.r.t the representative of clusters. We call each representative as the center of

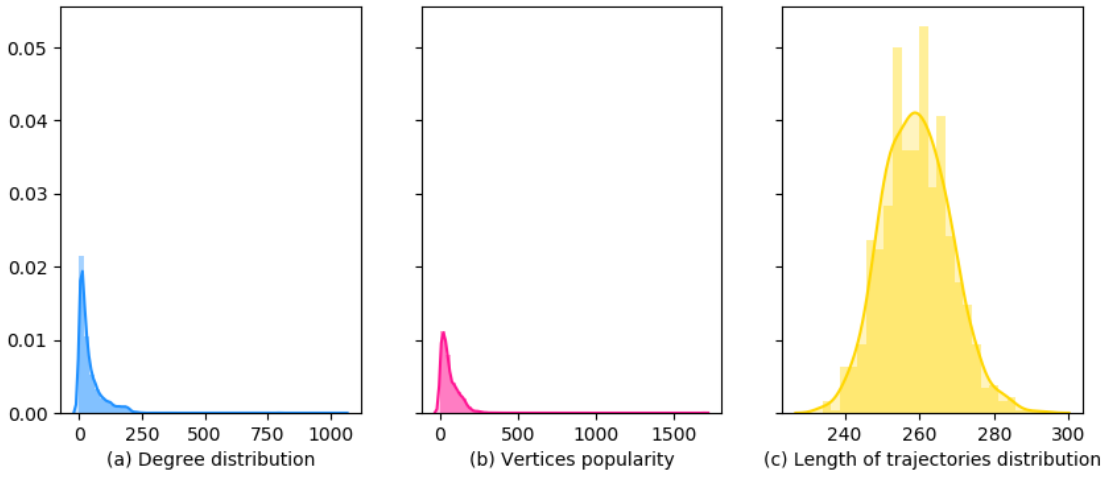


Figure 4.2: Properties of Facebook dataset: (a)The degree distribution, (b)The popularity of nodes, (c)The distribution of the length of trajectories

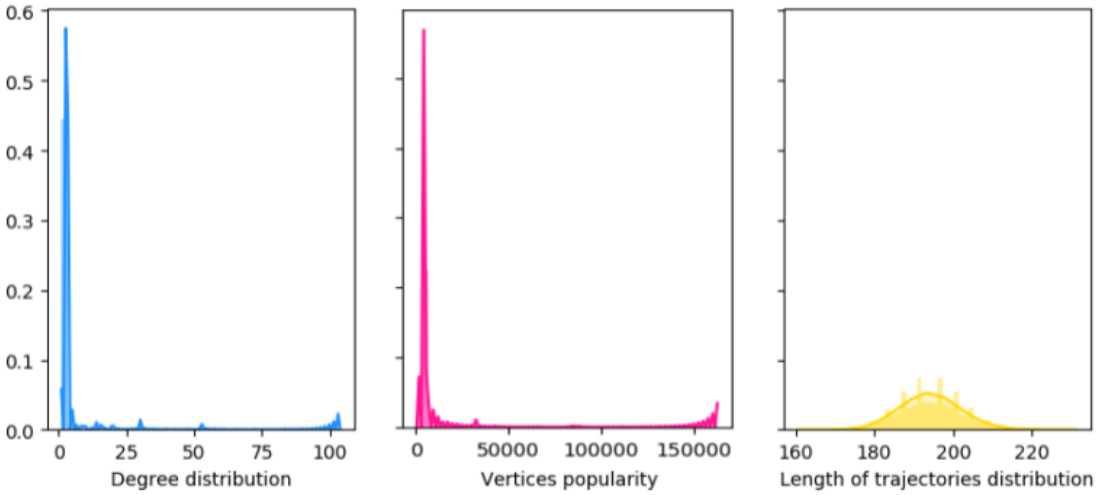


Figure 4.3: Properties of WEB-Network dataset: (a)The degree distribution, (b)The popularity of nodes, (c)The distribution of the length of trajectories

the cluster. As a result, we have a sequence of time-stamped centers assigned to user *id*, which is a trajectory for the user *id*. Finally, by traversing each trajectory of each user, we make a graph with the k-means centers as the

Dataset Name	#trajectories	#nodes	#edges	Diameter
Facebook Dataset	1000	4039	88234	8
WEB Network (WEBN)	100K	3,032	6,475	11
Milan Dataset	16166	3000	130071	5
Rome Dataset	7755	473	10524	6

Table 4.1: Summary of Datasets

nodes set. There is an edge ij between two centers i, j of the graph, if there exists at least one trajectory traversed through i, j, consecutively. The archived graph contains 3000 nodes and 130071 edges. Figure 4.4(a) shows the degree distribution of nodes of the graph. After mapping, each trajectory is represented in the form defined in Section 3.1. To sum up, give a sequence of time-stamped spatial points with at least two observations, we are able to build a trajectory by mapping each spatial point to the center of clusters.

The preprocessed dataset consists of 16166 trajectories. The average length of the trajectories in terms of the number of nodes is about 87. A summary of the dataset is given in the Table 4.1. Figure 4.4(b) reports the distribution of popularity of each node, where the popularity of a vertex is the number of trajectories passing through it, while Figure 4.4(c) shows the distribution of the length of the trajectories.

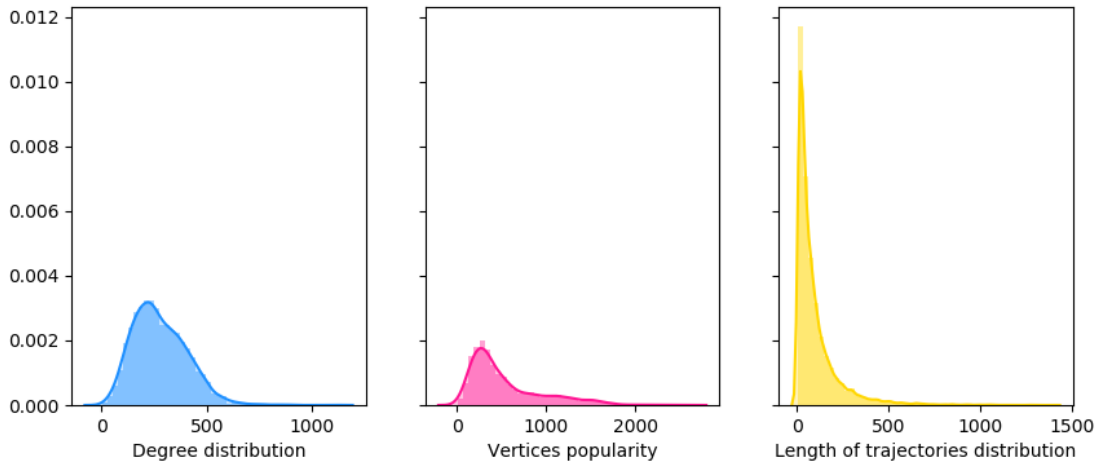


Figure 4.4: Properties of Milan dataset: (a)The degree distribution, (b)The popularity of nodes, (c)The distribution of the length of trajectories

Real Trajectory Set: Rome To evaluate our method we use a dataset provided in [60] containing tourist movements covering Rome. In this dataset, the set of Points of Interest (in short, PoIs) in each city is aggregated from Wikipedia. For building tourist trajectories, the geo-tagged photos from Flickr are collected and the photos are mapped to the set of PoIs aggregated from Wikipedia. Note that, since users may have taken more than one photo assigned to the same PoI, the timestamps of the first and the last photo are considered as the starting and ending time of their visits at the PoI in the dataset. In our experiments, by considering the timestamps of the first photo as the time to reach the PoI, we build the trajectory for each user as defined in Section 3.1. Then, by traversing each trajectory, we build the underlying graph, whose vertices are PoIs and whose edges correspond to transactions between two PoIs. The main properties of the network are shown in Table 4.1. Moreover, the Figure 4.5 shows the distribution of the node degrees on the network, the popularity of each node, and the distribution of the length of the trajectories.

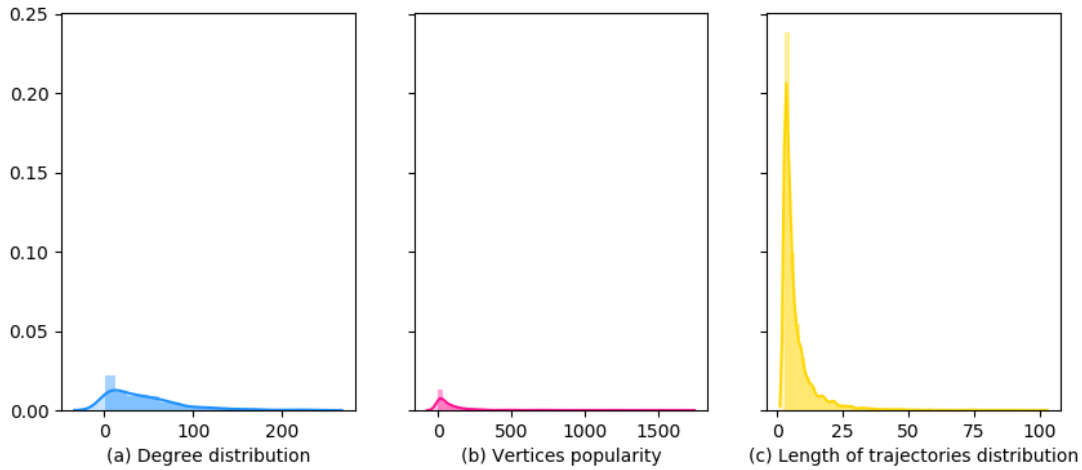


Figure 4.5: Properties of Rome dataset: (a)The degree distribution, (b)The popularity of nodes, (c)The distribution of the length of trajectories

4.2.2 Query Time

In the following, we compare the query time of the three methods. Table 4.2 reports our results, showing the average query time over 100 queries over trajectories on Milan and Rome. As it can be seen, both SHQ and SHQT variants outperform BASE. The most evident benefit can be seen for the

biggest dataset we considered i.e. Milan dataset. In this case, SHQT spends less than 23% of the time needed by BASE and SHQ.

Datasets	BASE	SHQ	SHQT
Milan	380.03	376.15	85.48
Rome	26.19	19.42	15.83

Table 4.2: The average time for answering a query for each proposed method on each dataset. For Facebook and WEB, refer to Figure 4.6.

We report in Table 4.3 the number of candidate trajectories for all the methods (i.e. $|\Gamma|$ and $|\tilde{\Gamma}|$). The table shows that SHQ and SHQT select more candidates than BASE. This is not an issue, as even if we have to process these extra trajectories, we process queries faster as shown in Table 4.2. This is due to that, by shrinking either only query or both query and target trajectories, we reduce the number of shortest path distance computations and consecutively, the query time will be less than BASE.

Datasets	BASE	SHQ	SHQT
Milan	9786.39	9968.98	9968.98
Rome	7504.37	6569.84	6569.84

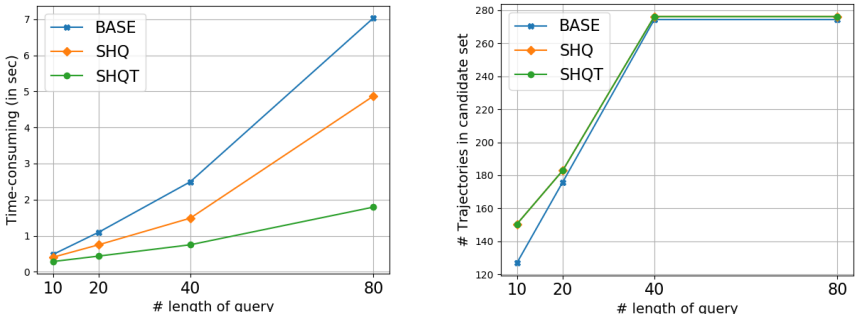
Table 4.3: The average number of trajectories in the candidate set in each method. For the Facebook dataset, refer to Figure 4.7.

For the sake of completeness, we have also analyzed the behavior of our method when the length of the query trajectories varies for the case of the Facebook and WEBN datasets.

Figure 4.6 shows the results of experiments on Facebook. As we can see in Figure 4.6(a), SHQ and SHQT outperform BASE. SHQT significantly outperforms SHQ and BASE and the improvement becomes even more evident when the length of the query trajectory increases. As the length of the query goes up to 80, the time needed by BASE and SHQ variants increases faster than the SHQT variant. It confirms that shrinking both target and query trajectories reduces the number of distance computations and thus the time reduced greatly.

Indeed, Figure 4.6(b) shows the average number of trajectories in the candidate set for each method. Note that the number of trajectories in the candidate set for both methods SHQ and SHQT is the same since we use the same approach for getting $\tilde{\Gamma}$. As can be seen, by increasing the length of

the query up to 40, the number of trajectories in the candidate set for each method increases quickly to more than 240 and then becomes the same for all of them. This confirms the role of the precomputed distances among Voronoi centers to accelerate query processing. The time cost of this precomputation is negligible. Figure 4.7 shows the results of the experiments on WEBN.

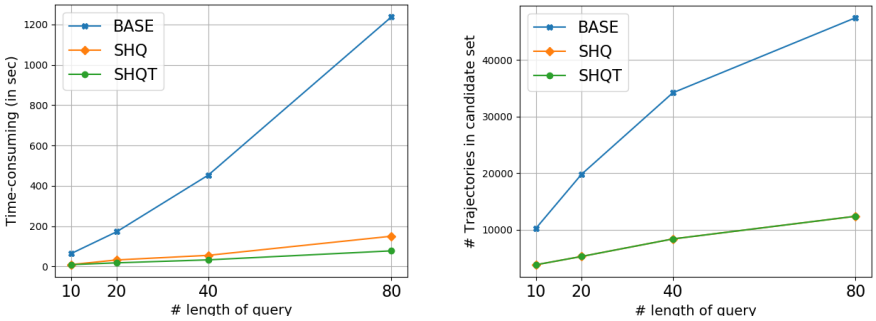


(a) (b)

Figure 4.6: (a) The average of running time and (b) the average number of trajectories in candidate set (i.e. the number of similarity score computations) by each method vs the different length of query trajectories on Facebook.

As shown in Figure 4.7(a) the running time of BASE is dramatically larger than SHQ and SHQT, while the difference between running time of SHQ and SHQT is not significant. This is due to that, the number of similarity score computations by each method is greatly less than BASE (see Figure 4.7(b)). This confirms that the Voronoi partitioning over larger set of trajectories on WEBN reduce the running time by reducing the number of similarity score computations rather than the number of shortest path distance computations. Considering those datasets containing huge set of trajectories, we can treat k-MsTraj with the exact approaches, efficiently, which we will study these methods in the next chapter.

In summary, the experiments on running time of the methods on each dataset show that the proposed methods outperform the baselines and provide strong evidence of the performance and robustness of our solutions.



(a) (b)

Figure 4.7: (a) The average of running time and (b) the average number of trajectories in candidate set (i.e. the number of similarity score computations) by each method vs the different length of query trajectories on WEBN.

4.2.3 Preprocessing Time

In order to analyzing the performance in comparison with the baselines, we tested the robustness of our approach by considering the precomputation cost for each datasets.

Considering both variants of the proposed method, the only difference between the preprocessing of SHQT and SHQ is that only the former uses the precomputed distances. In Table 4.4, we report the time needed to perform the indexing and shrinking the trajectories. In particular, columns NTrajI and VoTrajI report the time needed for building respectively the indexing structures NTrajI and VoTrajI. The column of "Distance Precomputing" shows that the time needed to precompute the distances is negligible. Finally, we can observe that the time needed to perform the Voronoi partitioning and for shrinking trajectories in the last column is also negligible with respect to the time needed for building NTrajI, which clearly dominates the cost. The results of experiments on each dataset confirms the robustness of our method, regarding the precomputation cost.

4.2.4 Quality Evaluation

We evaluate the quality of the solution produced by the SHQ and SHQT methods w.r.t BASE. The effectiveness of the method is assessed by means of the metrics that we describe next, where the values close to 1 are more desirable. Let γ_1 and γ_2 be two output sets containing top-k trajectories e.g.

Dataset	NTrajI	VoTrajI	Distance precomputing	Shrinking trajectories and building Voronoi diagram
Facebook	185.19	0.51	0.48	0.62
WEBN	4599.10	7.18	0.06	65.06
Milan	1716.44	13.50	0.27	10.21
Rome	69.25	0.24	0.005	0.56

Table 4.4: Preprocessing time (in sec.)

the exact and approximated solutions, respectively.

1. We define the *similarity score ratio* as the ratio of the average similarity scores of trajectories in γ_1 and γ_2 as follows:
$\text{SSR}(\gamma_1, \gamma_2) = \frac{\sum_{T \in \gamma_1} Sim(Q, T, t)}{\sum_{S \in \gamma_2} Sim(Q, S, t)}$.

2. We define the *intersection ratio* as $\text{IR}(\gamma_1, \gamma_2) = \frac{|\gamma_1 \cap \gamma_2|}{k}$.

Our results are shown in Figure 4.8 and 4.9, where the IR and SSR ratios are reported as a function of k. In particular, Figure 4.8 represents the IR ratio with increasing k on each datasets. The IR ratio goes up to more than 0.80 quickly, by increasing the value of k on the Facebook network. On the other hand, this value in Milan and Rome networks becomes close to 0.3. However, we observe that the lower values of IR correspond to SSR values that are close to 1. Indeed, Figure 4.9 shows SSR which is almost always very close to 1 and that gets more close to 1, while k is increasing from 1 to 64.

Moreover, as mentioned before, Voronoi partitioning shrinks trajectories in Facebook dataset more than trajectories in WEBN. Thus, we expect to have more accurate results in WEBN than Facebook network in terms of the IR ratio. Figure 4.8(b) confirms our exception.

Indeed, by Lemmas 6 and 7, we would expect that the similarity score would be larger when we shrink trajectories. Indeed, by results in Figure 4.10 we can observe that the similarity scores by shrinking trajectories behave as we expected.

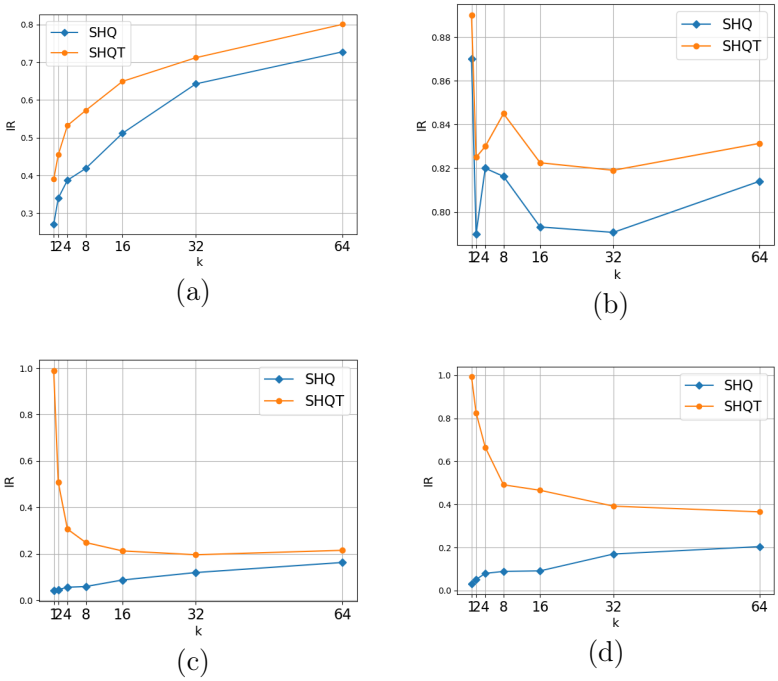


Figure 4.8: The quality of the results returned by the competitors in terms of IR ratio vs different values of k; (a)1K synthetic trajectories on Facebook; (b)100K synthetic trajectories on WEBN; (c) Real dataset on Milan; (d) Real dataset on Rome

4.3 Conclusion

In this chapter, we proposed an approximate method treating the k-MsTraj problem, with the aim of reducing the number of shortest path distance computations between two trajectories. In particular, we provided a comprehensive representation of trajectories in the dataset, by taking advantage of the network Voronoi partitioning. Thereafter, we estimated the similarity between two trajectories, by using the distance between Voronoi centers. We distinguished between two variants of the algorithm, corresponding to how the trajectories are shrunk during the query processing: SHQ, if only the query trajectory is shrunk, and SHQT, when both query and target trajectory are shrunk.

We evaluated the performance of the proposed approximate algorithm, regarding the time needed to process the query, the time needed to preprocess the trajectory set and the quality of the produced approximate solution, comparing our results with the baseline. We showed that our proposed methods

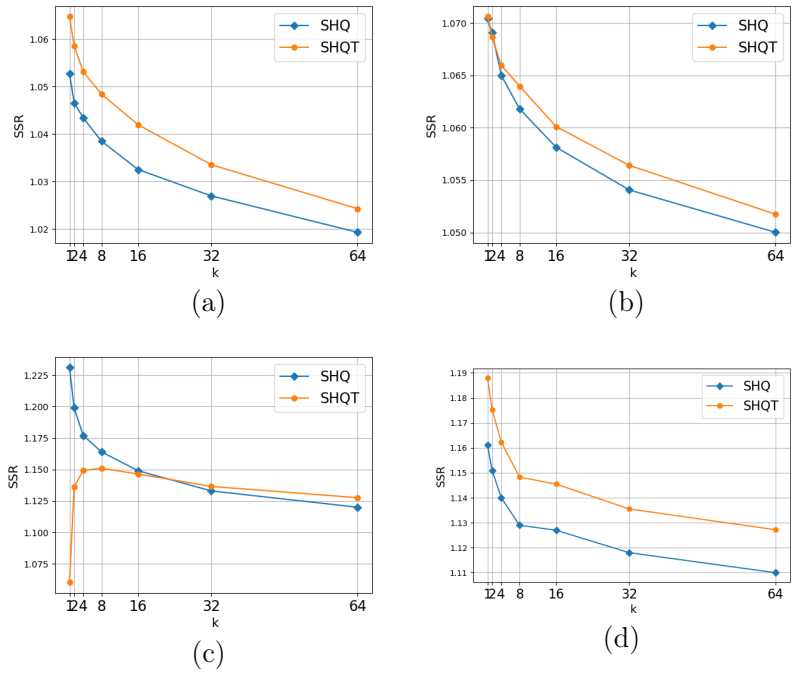


Figure 4.9: The quality of the results returned by the competitors in terms of SSR ratio vs different values of k; (a)1K synthetic trajectories on Facebook; (b) 100K synthetic trajectories on WEBN; (c) Real dataset on Milan; (d) Real dataset on Rome

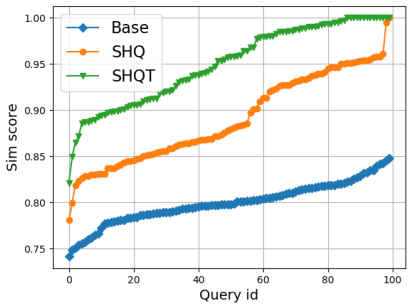


Figure 4.10: The comparison between similarity scores of each method on Facebook network.

are effective, by conducting experiments on two real datasets of movements in Milan and Rome, and two synthetic trajectory sets on real-network of Facebook and Web network.

The next evaluation takes into account the precision of the solutions provided by the approximated methods, in terms of two proposed metrics. The results showed that reasonably good solutions made by the methods, having the best performance for smaller values of k, which is natural in real cases.

The proposed method in this chapter is more efficient for treating datasets containing long trajectories on more dense networks. The results of this chapter can be used by those applications that concern fast results, even if the accuracy of the results is less than 100%. For example, analyzing the behavior of users on social networks, by aiming at clustering users based on their activities during the time can benefit from this study.

Chapter 5

Trajectory Similarity Search: Pruning Approach

In this chapter, we provide two exact approaches for answering the k-MsTraj query, that is, identifying the trajectories are most similar to a given query trajectory within a given time interval.

This kind of query typically needs to compute the shortest path distances between all trajectories in dataset to identify the most similar ones, which it could be expensive, facing with large trajectory set. To accelerate this processing, since computing the network distance (i.e. shortest path distance) between trajectories is expensive, a query processing algorithm aims at minimizing the number of distance computations. One approach is prune-and-refine method, which aims at computing the similarity score for a set of most promising trajectories in dataset instead of all trajectories. In particular, the pruning step makes a set of candidate trajectories, and the refine step identifies the actual results from the candidate set.

We propose two kinds of pruning algorithm. In particular, by making use of the indexing structure introduced in Section 3.2, we make the candidate set of trajectories by searching through each node of the query. In one proposed approach called Ordered-Based method, we aim at reducing the number of query nodes visited during the searching process. While, in another proposed approach, named SkyLine, we aim at making the pruning step, more effective, by visiting all nodes of the query. In this approach, we first make a candidate set of relevant trajectories, by searching through each node of the query. Then, we apply the second step of pruning, to find the most relevant trajectories. Indeed, by using this approach, we also provide an approximate method, which enables us to find the top-k most similar trajectories to the given one, fast and accurate.

Properties	PTM [79]	This thesis
Query	One trajectory	One trajectory
Output	Most similar trajectories	Most similar trajectories
Network	**Spatial**	**Topological**
Trajectory	A sequence of **timestamped** nodes	A sequence of nodes with **the corresponding time intervals** the trajectory spending on each node
Similarity function	**Linear combination** of spatial distance and temporal distance, both are LCSS-based	The **Network distance** between two nodes of two trajectories is **combined with the time interval** that each trajectory spent on the nodes
Optimization technique	Pruning trajectories **spatially**, by network expansion Pruning trajectories **temporally**, by 1-D range searching on a set of timestamp points Pruning spatially-temporally **independently**	Pruning trajectories based on the time intervals the trajectories spent on each node and their location on the graph, in a **single searching process**

Table 5.1: Method comparison

The most related work to the study of this project is arguably the PTM query [79], which takes the spatiotemporal similarity into account. We cover the differences between PTM and the proposed problem in this chapter in Table 5.1. Duo to these differences, PTM and our problem call for different algorithms for query processing. In PTM they search by each node of query for closeby trajectories in the spatial and temporal domain, independently. This is due to that, their similarity function also computes spatial and temporal distances, independently, with the quadratic number of network distance computations, as discussed in Chapter 2. The PTM uses the network expansion [26] for finding trajectories that are spatially close to each node of the query. Then, by considering all timestamps of trajectory nodes on a time axis, they find those trajectories have a timestamp within a specified range from each query timestamp. Instead, we index trajectories w.r.t the time intervals that they spent on each node. Therefore, our method finds close trajectories to each query node by considering the temporal aspect in a single searching process. Due to these differences, the PTM method and other similar works (e.g [82, 74, 75]) do not work for the proposed problem in this thesis.

This chapter is organized as follows. First, in Section 5.1.1 we propose the Ordered-Based technique aim at visiting less number of query nodes during the searching process, and we discuss two variants of this method. Then, in Section 5.1.2 we present the SkyLine method based on dominance relationships. Finally, we develop an approximation technique that is more efficient than the exact counterpart but the accuracy of the results may be less than 100% in Section 5.1.3. We evaluate the proposed methods by conducting

comprehensive experiments over real-world network datasets in Section5.2. Due to the lack of competitors, as discussed earlier, we evaluate the effectiveness of the proposed methods in comparison with the BASELINE method designed in Section 3.3. The performance evaluation results demonstrate that the proposed techniques are efficient and scalable.

5.1 Incremental Pruning Techniques

Given a set of trajectory $\mathcal{T}$ on graph G, a query Q and time interval T, we aim at addressing the K-MSTRAJ problem. Let $\mathcal{X} = (G, \mathcal{T}, Q, t = [a, b], k)$ be an input for the K-MSTRAJ problem. Given an input $\mathcal{X}$, if we can find a subset $\mathcal{T}' \subset \mathcal{T}$ of k trajectories such that $min_{T \in \mathcal{T}'}\{Sim(Q, T, t)\} \geq max_{S \in \mathcal{T}-\mathcal{T}'}\{Sim(Q, S, t)\}$, then based on the Definition 4, the subset $\mathcal{T}'$ is K-MSTRAJ.

Theorem 2 *For a given input $\mathcal{X} = (G, \mathcal{T}, Q, t = [a, b], k)$ for the* K-MSTRAJ *problem, if there exists a set $\mathcal{T}'$ containing k trajectories in $\mathcal{T}$ such that $LB(\mathcal{T}') \geq UB(\mathcal{T} - \mathcal{T}')$, then $\mathcal{T}'$ is* K-MSTRAJ.

Proof 5 Assume for each trajectory $S \in \mathcal{T} - \mathcal{T}'$, instead of computing the exact value $Sim(Q, S, t)$, we are able to estimate it as the best score value that S can achieve. We call this value as an upper bound $UB(S)$ for trajectory S w.r.t $\mathcal{X}$. Obviously, for each $S \in \mathcal{T} - \mathcal{T}'$ we have:

$$UB(S) \geq Sim(Q, S, t) \tag{5.1}$$

$$\max_{S \in \mathcal{T}-\mathcal{T}'}\{UB(S)\} \geq \max_{S \in \mathcal{T}-\mathcal{T}'}\{Sim(Q, S, t)\} \tag{5.2}$$

Thus, if we can find a set like $\mathcal{T}''$ such that:

$$\min_{T \in \mathcal{T}''}\{Sim(Q, T, t)\} \geq \max_{S \in \mathcal{T}-\mathcal{T}''}\{UB(S)\}$$

$$\xrightarrow{(5.2)} min_{T \in \mathcal{T}''}\{Sim(Q, T, t)\} \geq max_{S \in \mathcal{T}-\mathcal{T}''}\{Sim(Q, S, t)\}$$

Then the set $\mathcal{T}''$ would be K-MSTRAJ.
Assume, $LB(\mathcal{T}'') = min_{T \in \mathcal{T}''}\{Sim(Q, T, t)\}$ and $UB(\mathcal{T}-\mathcal{T}'') = max_{S \in \mathcal{T}-\mathcal{T}''}\{UB(S)\}$.
The proof of this theorem derives from the above when $\mathcal{T}''$=$\mathcal{T}'$. ■

We take the benefit of this theorem to design an *incremental pruning* technique (IP) to restrict the search space by two different methods described in Sections 5.1.1, 5.1.2.

Basic Idea The basic idea of the incremental pruning technique is as follows. The algorithm moves along a sequence of steps (the steps depend on the method) and finds a set of candidate trajectories at each step. The algorithm uses a general heap of size of k to maintain k trajectory ids with the highest scores during the searching process. The values in the heap make a solution for k-MsTraj, which is built incrementally while adapting to the changes of candidate trajectories at each step. The algorithm determines the lower and upper bounds of the similarity score of trajectories at each step. In particular, the lower bound (LB) at each step is the minimum similarity score of the trajectories in the heap. Besides, the upper bound (UB) would be the best similarity score of those trajectories that have not discovered during the current search steps. The value of UB is defined base on the method. The termination condition is defined based on the Theorem 2. Therefore, the algorithm halts when $UB \leq LB$.

In the next, we design two Ordered based and the SkyLine methods to build a solution for the k-MsTraj problem, efficiently. We specify for each method, the *steps*, the *candidate trajectories* at each step, and the *upper bound* in the incremental pruning technique.

5.1.1 Ordered Based Method

In this method, the IP algorithm moves across the nodes of the query and keeps the heap including the top-k trajectories updated at each node of the query. In particular, the nodes of the given query trajectory are the steps for the IP algorithm. For a given query Q and time interval t, we first restrict query within t as $Q[t]$. Then, for each $(v_i, t_i) \in Q$, we use the indexing described in Section 3.2 and by searching over IT_{v_i} we find $\Gamma_{(v_i,t_i)}$ as a candidate trajectories at each node. For each trajectory in $\Gamma_{(v_i,t_i)}$, we compute the similarity score with respect to the function in Definition 3. We build the heap H containing top-k trajectories while updating during the search process.

For a given input $\mathcal{X}$, the Ordered method aims at achieving the k-MsTraj by reducing the number of visited nodes of the query during the search process (i.e. steps for IP) and the number of computing similarity scores at each step (i.e. the candidate trajectories ate each step for IP). To this end, the algorithm follows the two-variant ordering of query nodes:

Count-Based Ordered (CBO): Sorting nodes of query w.r.t the number of trajectories passing through the node.

Time-Based Ordered (TBO): Sorting nodes of query w.r.t the number

of trajectories passing through the node within its corresponding time interval.

In the first case, we can simply assign the number of trajectories passing through each node in V, in the preprocessing operation. Therefore, in the query processing, for a given trajectory Q with the length of l, we only sort the nodes of Q w.r.t the values are assigned to them in $O(l \times \log l)$ time.

For the second case, since the number of trajectories for each node depends on the time intervals are assigned to them, we cannot calculate them in the preprocessing. Therefore, in the query processing, we first restrict query Q within time interval t. Then for $(v_i, t_i) \in Q[t]$, we search over the IT_{v_i} by NTrajI-search(v_i, t_i) to find $\Gamma_{(v_i,t_i)}$. We sort the nodes of $Q[t]$ w.r.t $|\Gamma_{(v_i,t_i)}|$.

Algorithm 3 shows the general framework of the Ordered Based Method. Regarding CBO and TBO, the only difference is the order of nodes of the query are visited. Generally, CBO will find more candidate trajectories, efficiently, while TBO finds most promising trajectories as a candidate, by consuming more time to put query nodes on the time-based order. We show in our experiments how involving the temporal aspect of trajectories on query nodes ordering (i.e. TBO) affects the performance.

Now, the incremental pruning algorithm moves across the nodes of the ordered query Q^o, making heap H. As illustrated in Algorithm 3, the algorithm searches over the IT_{v_i}, for each $(v_i, t_i) \in Q^o[t]$ to build $\Gamma_{(v_i,t_i)}$. By computing the similarity score for each trajectory in $\Gamma_{(v_i,t_i)}$, we update heap H.

Let $|Q^o[t]| = l$ and $1 \leq i \leq l$ and H_i containing k trajectory ids with highest similarity scores have discovered by visiting i nodes of $Q^o[t]$. We determine the lower bound as the minimum similarity score of trajectories in H_i as $LB_i = \min_{T \in H_i}\{Sim(Q, T, t)\}$. On the other hand, we determine the upper bound as the best similarity score of those trajectories that have not discovered within i steps of the searching process. Note that, for a pair $(v_i, t_i) \in Q^o[t]$ if a trajectory T is not discovered by searching over IT_{v_i}, we have $dist(v_i, T, t_i) \geq 2$.

Considering those trajectories like T that they have not discovered by searching process through i nodes of query, since the upper bound is a best similarity score for them, so for each visited node $(v_z, t_z) \in Q^o[t]$ for $1 \leq z \leq i$, we consider $dist(v_z, T, t_z) = 2$ and for the remained nodes $(v_w, t_w) \in Q^o[t]$ for $i < w \leq l$ we have $dist(v_w, T, t_w) = 0$. Therefore, at each $(v_i, t_i) \in Q^o[t]$, we define the upper bound as follows.

Algorithm 3: CBO & TBO

Input: Graph G, set of trajectories $\mathcal{T}$, query trajectory Q, time interval $t = [a, b]$, integer k
Result: k-MsTraj

H /* H is a heap in size of k */
upper bound UB and lower bound LB /* UB, LB are dynamic and update during the searching process */
$Q[t]$ /* restricted Q within t */
$Q^o[t] \leftarrow$ sort($Q[t]$) /* sort method is either CBO or TBO */
while $heap - size(H) < k$ **do**
    $\Gamma_{(v_i,t_i)} \leftarrow NTrajI - search(v_i, t_i)$ for $(v_i, t_i) \in Q^o[t]$
    $H \leftarrow$ trajectories in $\Gamma_{(v_i,t_i)}$
    update UB
$LB \leftarrow Sim(Q, min(H), t)$ /*min(H) returns a trajectory id in H with minimum score */
while $LB < UB$ **do**
    $\Gamma_{(v_i,t_i)} \leftarrow NTrajI - search(v_i, t_i)$ for not visited $(v_i, t_i) \in Q^o[t]$
    for $T \in \Gamma_{(v_i,t_i)}$ **do**
        if $Sim(Q, min(H), t) < Sim(Q, T, t)$ **then**
            H.remove(min(H))
            H.add(T)
    $LB \leftarrow Sim(Q, min(H), t)$
    Compute UB
k-MsTraj $\leftarrow$ trajectories in H

$$UB_i = \frac{\sum_{\substack{(v_z,t_z)\in Q^o[t] \\ 1\le z\le i}} |t_z| \times e^{-2/D_G} + \sum_{\substack{(u_w,t_w)\in Q^o[t] \\ i<w\le l}} |t_w|}{|t|} \tag{5.3}$$

The values of the upper and lower bounds are dynamic and continuously change at each $(v_i, t_i) \in Q^o[t]$ during the searching process. Based on the Theorem 2, the algorithm stops saerching at step $i-$th and report trajectory ids in H_i, if $LB_i \geq UB_i$ (see Algorithm 3).

5.1.2 SkyLine Method

In this section, we aim at solving the k-MsTraj problem by a fewer number of similarity score computations. To this end, we first, use the candidate set Γ described in the BaseLine method is Section 3.3. Although, the additional information is assigned to each discovered trajectory in Γ to make the steps in IP technique as explained in Section 5.1.2. Then, we describe how the SkyLine method prunes the trajectories in Γ, incrementally, by determining the LB and UP at each step in this section.

Making Steps in IP technique

We make a candidate set similar to the BaseLine method with annotating useful information to each trajectory. First, we make the query trajectory Q restrict within the time interval t. Then, we move along the nodes of $Q[t]$, searching for trajectories cross the nodes within the corresponding time interval. For each $(v_i, t_i) \in Q[t]$, we search over IT_{v_i} to make $\Gamma_{(v_i,t_i)}$. For each reported trajectory id $T \in \Gamma_{(v_i,t_i)}$ and the corresponding time interval t_j (i.e. there exist a vertex $u_j \in \{v_i\} \cup N(v_i)$ and $(u_j, t_j) \in T$ and $t_i \cap t_j \neq 0$), we can easily drive $dist(v_i, T, t_i)$ as follows.

$$dist(v_i, T, t_i) = \begin{cases} 0, & \text{if } \exists i \in t_i | T(i) = v_i \\ 1/D(G), & \text{otherwise} \end{cases} \tag{5.4}$$

Therefore, once we traverse all nodes of the query, a lower bound LB of similarity for each trajectory T in Γ can thereafter be computed by using the known distances defined in Equation 5.4.

$$LB(T) = \frac{\sum_{(v_i,t_i)\in Q[t] | T\in\Gamma_{(v_i,t_i)}} |t_i| \times e^{-dist(v_i,T,t_i)}}{|t|} \tag{5.5}$$

Moreover, for each reported trajectory $T \in \Gamma_{(v_i,t_i)}$ with its corresponding time interval t_j, which $t_j \cap t_i \neq 0$, we are able to measure the length of the

time interval spent by T either in v_i or its neighbors, where $v_i.t(T) = |t_j \cap t_i|$. Therefore, we determine the whole time that T spent close to the query Q within t as the *time bound* of T as follows:

$$TB(T) = \sum_{(v_i,t_i)\in Q[t]|T\in\Gamma_{(v_i,t_i)}} v_i.t(T) \tag{5.6}$$

On the other hand, if there exists a pair $(v,t) \in Q[t]$ such that $T \notin \Gamma_{(v,t)}$, it means there is no time instance $i \in t$ such that $T(i) \in v \cup N(v)$. Thus, $dist(v,T,t) \geq 2$. Therefore, the minimum distance that a trajectory outside the $\Gamma_{(v,t)}$ can achieve, is the value of 2. We define, an upper bound of similarity for a trajectory in Γ as follows:

$$UB(T) = LB(T) + \frac{\delta \times e^{(-2)/D_G}}{|t|} \tag{5.7}$$

where $\delta = |t| - |TB(T)|$.

By using these bounds, we can set up a pruning mechanism for the k-MsTraj to avoid computing the similarity score for the whole trajectories in Γ and thus restrict the search space more.

To answer the k-MsTraj problem, the trajectories in Γ with larger lower bound and time bound are supposed to be similar to the query with a high similarity score. Based on this basic idea, we propose the SkyLine algorithm, which retrieves the most similar trajectories w.r.t the query. Before solving the problem, we first transform each trajectory $T \in \Gamma$ with two values $LB(T)$ and $TB(T)$ to a 2-D point $p_T(x_T, y_T)$ on the plane, such that $x_T = TB(T)$ and $y_T = LB(T)$. Figure 5.1(a) shows a representative example of transformed trajectories in candidate set by considering one query trajectory randomly chosen in the Oldenburg dataset (See Section 5.2 for more details about this dataset). Let P be a set of $|\Gamma|$ points on the plane, one per trajectory in Γ. We explain how the SkyLine method uses this set of points to find k-MsTraj as follows.

SkyLine Query Processing

Given a set P of 2-D points, one for each trajectory in Γ, we aim at finding those points that have a maximum x and y coordinates. Here x, y show the lower bound(LB) and time duration(TB) of each trajectory in Γ, respectively. It is equivalent to finding the *Pareto frontier* of a collection of points. To answer k-MsTraj when $k = 1$, we can find the Pareto frontier of the points in P and then the most similar trajectory is supposed to be on the Pareto frontier. For larger values of k, we may need to compute a sequence of the

Pareto frontiers as Pareto layers. As shown in Figure 5.1(b) the outermost one is the first Pareto layer and the rest are formed in the same way, recursively. The outermost layer may be degenerate, consisting only of one or two points that we expected to be the most similar one. We will evaluate this observation in our experiments. By using these Pareto layers, we provide the incremental pruning strategy including the upper and lower bound of similarity of trajectories in each Pareto layer (as described in Section 5.1). In particular, each Pareto layer of the points in P would be one step in the IP technique. Indeed, the trajectories lying on each Pareto layer would be the candidate trajectories at each step of the IP technique.

Assume L is a Pareto layer containing a set of points with the corresponding trajectory ids. We define an upper bound for similarity score of trajectories lying on the layer L as $UB(L) = \max_{T \in L}\{UB(T)\}$.

To answer k-MsTraj problem, as illustrated in Algorithm 4, the Skyline method creates incrementally the Pareto layers. At each layer, we compute the similarity score for those trajectories lying on the layer to keep updated the heap H in the setting of the IP technique. So, at each layer L_i, we define the lower bound as $LB_i = \min_{T \in H_i}\{Sim(Q, T, t)\}$. When H_i adapted to changes at layer L_i. The algorithm stops searching on the layer L_i and reporting trajectory ids in H_i if $UB(L_{i+1}) \leq LB_i$ (see Algorithm 4).

We will show the correctness of the algorithm in Theorem 3.

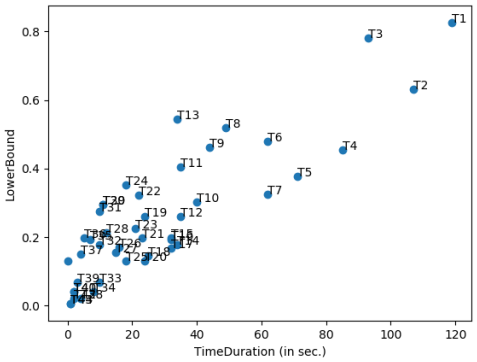
Lemma 8 $UB(L_{i+1}) \leq UB(L_i)$

Proof 6 Let T and S are two trajectories with maximum upper bound on two Pareto layers L_i and L_{i+1}, respectively. By the definition of Pareto frontiers, the point $p_T(TB(T), LB(T))$ dominates the point $p_S(TB(S), LB(S))$. So, $TB(S) \leq TB(T)$ and $LB(S) \leq LB(T)$. Therefore, by the definition of upper bound of a trajectory in Equation 5.7, we can drive that $UB(S) \leq UB(T)$. Then $UB(L_{i+1}) \leq UB(L_i)$. ■

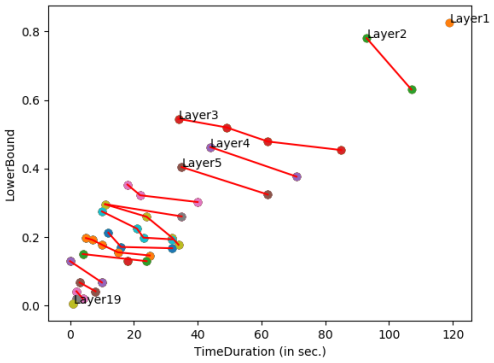
Theorem 3 *Given a set of trajectories $\mathcal{T}$, a query trajectory Q, and a query time interval t, if we can get a set of k trajectories maintained in heap H by searching over a sequence of i Pareto layers of transformed candidate trajectories to 2-D points on the plane, such that $UB(L_{i+1}) \leq LB_i$, then the heap H contains the* k-MsTraj.

Proof 7 For each trajectory $T_{i+1,j}$ for $j = 1, \cdots, |L_{i+1}|$ on the L_{i+1} layer, it holds that:

$$Sim(Q, T_{i+1,j}, t) \leq UB(T_{i+1,j}) \leq UB(L_{i+1}) \tag{5.8}$$



(a)



(b)

Figure 5.1: (a)The set of transformed points of each trajectory $T_i \in \Gamma$; (b)Pareto layers

For each trajectory $T \in H$, we have also:

$$LB_i \leq Sim(Q, T, t) \tag{5.9}$$

Therefore, if $UB(L_{i+1}) \leq LB_i$ then two inequalities 5.9 , 5.8 can be combined using the transition property as follows:

$$Sim(Q, T_{i+1,j}, t) \leq UB(T_{i+1,j}) \leq UB(L_{i+1}) \leq LB_i \leq Sim(Q, T, t) \tag{5.10}$$

So, we have $Sim(Q, T_{i+1,j}, t) \leq Sim(Q, T, t)$, which depict the fact that all trajectories in the L_{i+1} layer have smaller than or equal similarity score to the trajectories on H. ■

Consequence 1 *By Lemma 8, we can derive that* $UB(L_z) \leq UB(L_{i+1})$ *for* $z > i+1$. *So,* $UB(L_z) \leq UB(L_{i+1}) \leq LB_i$ *and any subsequent layer* z, *by using the mathematical strong induction will definitely not containing any trajectory of the* K-MsTraj *set.*

Algorithm 4: SKY

Input: Graph G, set of trajectories $\mathcal{T}$, query trajectory Q, time interval $t = [a, b]$, integer k
Result: K-MsTraj

H /* H is a heap in size of k */
$\Gamma_{(v_i,t_i)} \leftarrow$ NTrajI-search(v_i, t_i) for each $(v_i, t_i) \in Q[t]$
$\Gamma = \bigcup_{(v_i,t_i)\in Q[t]} \Gamma_{(v_i,t_i)}$
Transform each $T \in \Gamma$ to a point $p_T(TB(T), LB(T))$ on the plane
while $heap - size(H) < k$ **do**
    $L_i, L_{i+1} \leftarrow$ Compute Pareto layers
    H.add(T) for each $T \in L_i$
$LB_i \leftarrow Sim(Q, min(H), t)$ /*min(H) returns a trajectory id in H with minimum score */
while $UB(L_{i+1}) > LB_i$ **do**
    $L_i, L_{i+1} \leftarrow$ Compute Pareto layers
    for $T \in L_i$ **do**
        if $Sim(Q, min(H), t) < Sim(Q, T, t)$ **then**
            H.remove(min(H))
            H.add(T)
    $LB_i \leftarrow min(H)$
K-MsTraj $\leftarrow$ trajectories in H

5.1.3 Approximate SkyLine (ApSky)

To answer the K-MsTraj problem, computing the similarity score between a query and a set of trajectories is unavoidable. Each of the proposed methods in this chapter aimed to solve the K-MsTraj problem by computing the minimum number of similarity scores to accelerate query processing. It is obvious that the number of similarity score computations by each method is more than k. In this section, we aim at making use of the SkyLine method to provide an approximated solution for the K-MsTraj problem, by computing only $O(k)$ number of similarity scores.

Given an input $\mathcal{X} = (G, \mathcal{T}, Q, t, k)$ for the k-MsTraj problem, we approximate top k trajectories, efficiently, with a good precision.

To this end, we first, make the set P of 2-D points on the plane of a set of candidate trajectories as described in Section 5.1.2. Then, the approximate algorithm finds the Pareto layers. In our approximate algorithm, the trajectories lying on the outermost layer have the highest similarity scores and those lying on the innermost layer have the minimum similarity scores. Therefore, the algorithm reports k trajectories lying on the Pareto layers, started from the outer layers to the inner layers. For example in Figure 5.1, let $k = 3$, the algorithm by visiting two Pareto layers makes an approximate solution for 3-MsTraj. We will evaluate the quality of the approximated solution by our experiments in Section 5.2.3.

5.2 Performance Evaluation

In this section, we present performance evaluation results showing the efficiency of the proposed algorithms. First, we provide a series of experiments to evaluate the performance of exact algorithms: BASE (the BaseLine method described in 3.3), CBO (count-based ordering), TBO (time-based ordering), and SKY (the SkyLine method), in Section 5.2.2. Then, another series of results follows demonstrating the performance of the approximate skyline-based algorithm (ApSky) in comparison with SKY, in Section 5.2.3 . All algorithms have been implemented in Python 3.

5.2.1 Network and Trajectory Datasets

Three real-world networks have been used for generating synthetic trajectories: *i*) the road network of the city of Oldenburg (OLN) in Germany with 5,835 nodes, *ii*) the road network of the state of California (CFN) in USA with 21,048 nodes and *iii*) a Web network (WEBN) [1] with 3,032 nodes. The main properties of the networks and dataset are represented in Table 5.2.

We have generated different trajectory datasets over each network, as shown in Table 5.2, by using the generator shown in Algorithm 2. The maximum duration is set to 30 and the maximum life span, i.e., the number of timestamps a trajectory is active, is set to 4,000.

Moreover, we have generated a set of 100K synthetic trajectories on the Oldenburg network using the well-known Brinkhoff's trajectory generator [2] to

[1] `http://networkrepository.com/web-edu.php`

[2] `https://iapg.jade-hs.de/personen/brinkhoff/generator/`

Network	#Nodes	#Edges	Diameter	#Trajectories
Oldenburg Network (OLN)	6,105	7,029	104	100K, 1M
California Network (CFN)	21,048	21,693	728	400K, 4M
Web Network (WEBN)	3,032	6,475	11	100K

Table 5.2: Summary of network and trajectory datasets.

evaluate the performance of the algorithms over different types of generated trajectories, described in Section 5.2.4.

5.2.2 Comparison of Exact Algorithms

The proposed algorithms are compared using different performance measures. In particular, we study the effectiveness of the proposed methods regarding the cost of each method per query as:

1. Running time: In particular we measure the time needed for answering the query by each method.

2. Number of similarity score computations: That is the number of candidate trajectories have discovered by each method, which has a direct effect on the running time in each one.

3. Number of visited nodes: In particular, the effectiveness of the Ordered based methods of CBO and TBO depends on the number of query nodes processed by each method; thus, we measure this value for each of these two methods.

The different parameters that affect performance are the number of results k, the length of the query trajectory, over different underlying networks. Table 5.3 presents the parameters and their corresponding values that have been used in our experiments. The results presented correspond to the average of 100 random queries executed in each case. For each query in our experiments, we consider the query time interval t in the setting of the Definition 4 as the whole lifetime of the query trajectory.

We evaluate the effectiveness of the proposed algorithm considering the different number of trajectories over each network. We conduct experiments on 100k and 1M trajectories on OLN, 400K trajectories on CFN, and 100k trajectories on WEBN. We consider two different sizes of trajectory datasets on OLN to show how the effectiveness of the methods could change over larger trajectories.

Table 5.3: Performance evaluation parameters.

Parameter	**Values**
Number of results (k)	1, 2, 4, 8, 16, 32, 64
Length of query (l_Q)	10, 20, 40, 80
Number of trajectories ($\|\mathcal{T}\|$)	100K, 400k, 1M, 4M
Network	OLN, CFN, WEBN

Different Values of k

The value of k in the k-MsTraj problem is a critical parameter that can affect the performance of the methods. In this part, we evaluate the quality of each method by varying the k values. We show the results in Figures 5.2–5.7 with the length of the query fixed to 20.

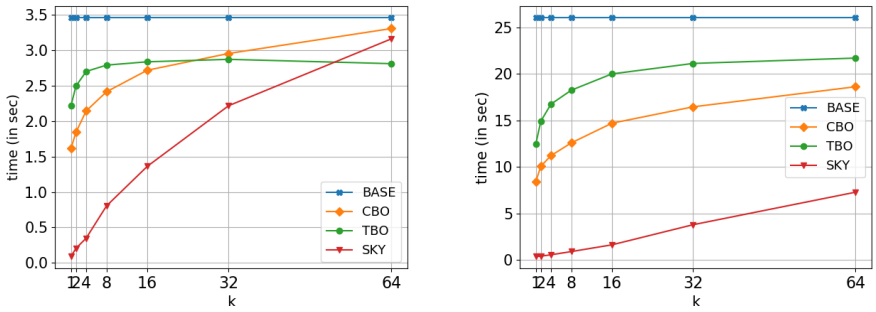


Figure 5.2: Evaluation of the proposed method regarding running time in comparison with BASE vs k on OLN (a) Experiments over 100k trajectories; (b) Experiments over 1M trajectories

As shown in Figure 5.2 and 5.3, the proposed methods outperform the BaseLine method considering running time and the number of score computations over two sets containing 100k and 1M trajectories on Oldenburg network. Note that, the cost of the BaseLine method is constant by varying the values of k since the BaseLine algorithm provided the solution of the k-MsTraj problem independently of the value k. In particular, considering 100k trajectories on OLN (see Figures 5.2(a) and 5.3(a)), we observe that SKY notably outperforms BASE and other two proposed methods (TBO and CBO) as the values of k goes up to 32. In which that the running time and the number of similarity score computations of the TBO and CBO methods increase to almost 3s and 350, respectively, while SKY increases costs with

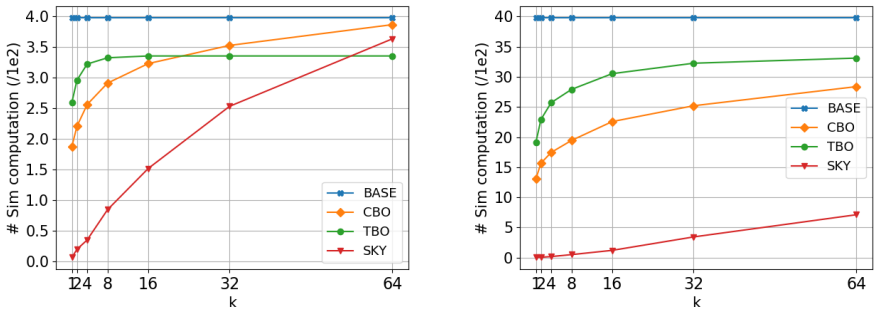


Figure 5.3: Evaluation of the proposed method regarding the number of similarity score computations in comparison with BASE vs k on OLN (a) Experiments over 100k trajectories; (b) Experiments over 1M trajectories

running time and number of score computations to more than 2s and 250, respectively. We observe that for larger values of $k = 64$, the TBO method outperforms SKY, while as Figures 5.2(b) and 5.3(b) show, by increasing the number of trajectories in dataset $\mathcal{T}$ from 100k to 1M over Oldenburg network, SkyLine outperforms other methods in running time and number of score computations, significantly, while k goes up to 64.

Regarding CBO and TBO, we observe in Figures 5.2(a) and 5.3(a) that CBO outperforms TBO for smaller values of k while considering larger number of trajectories as 1M, we see that CBO processes the query by computing fewer number of similarity scores and consecutively, consuming less time than TBO. This is because of that visiting the query nodes in order of the CBO method, visits more nodes of the query by increasing k values as shown in Figure 5.4, while computes fewer similarity scores at each query node and consecutively consuming less time, in comparison with TBO. However, TBO by processing more trajectories at primitive nodes of the query during the searching process increases the lower bound of trajectories and makes the Theorem 2 earlier to be satisfied.

Studying the behavior of 400k trajectories on CFN, Figure 5.5 shows the best performance of SKY in comparison with other methods. As shown in Figure 5.5(a), SKY outperforms all other methods by increasing the k values from 1 to 64. This is due to that, SKY process fewer number of trajectories during its searching process (see Figure 5.5(b)), while other methods process more trajectories and consecutively consume more time to answer the query while outperforming the BaseLine method. Comparing CBO and TBO,

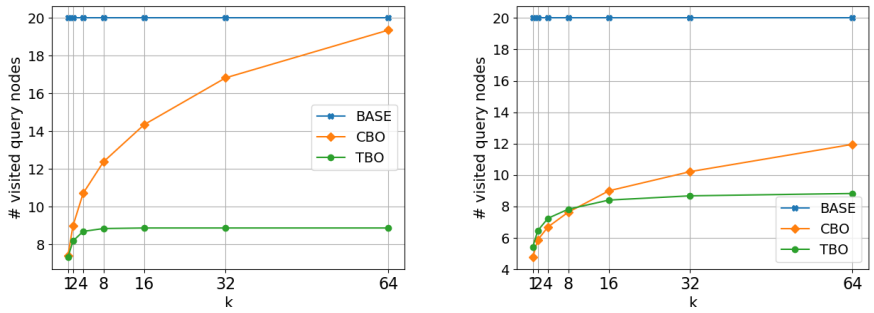


Figure 5.4: Evaluation of the performance of CBO and TBO w.r.t BASE in terms of the number of visited query nodes by varying the values k on OLN . (a) Experiments over 100k trajectories; (b) Experiments over 1M trajectories.

we observe that TBO behaves better than CBO for larger values of k, while CBO outperforms TBO for smaller k values, in terms of both number of score computations and consecutively running time. It is because of the similar reason mentioned earlier on the experiments over OLN.

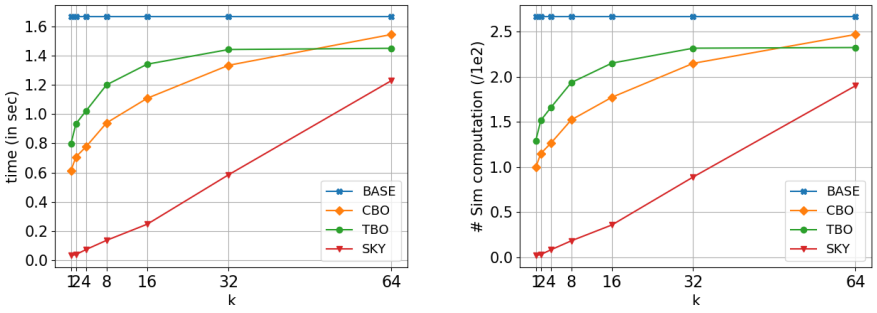
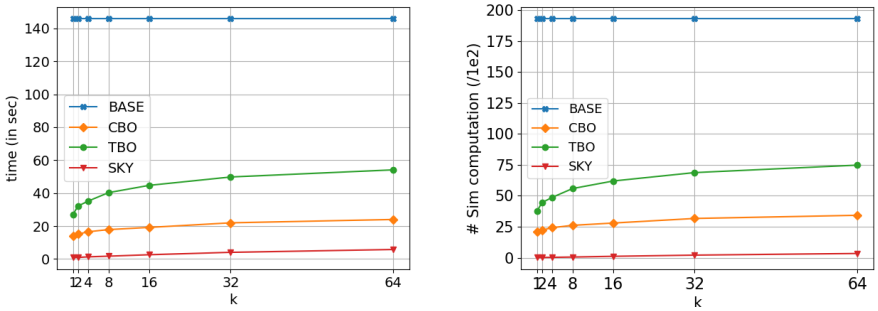


Figure 5.5: The results of experiments over 400k trajectories on CFN. (a) Running time in each method vs different values of k; (b) Number of similarity score computations by each method vs k.

Regarding the Web dataset, as shown in Figure 5.6, we observe that the total proposed methods outperform the BaseLine method. We can see that as the values of k go up to 64, the running time and the number of

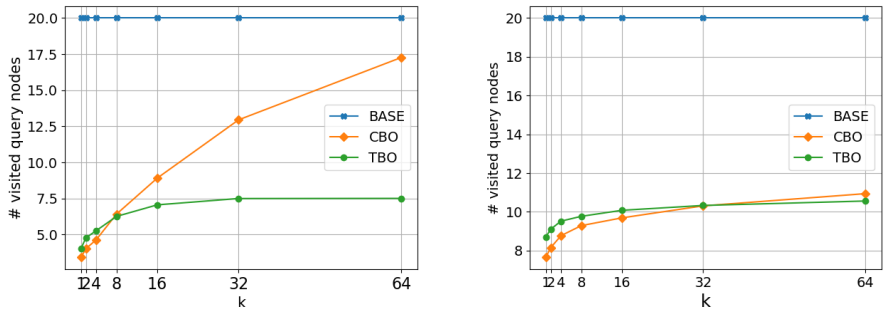
score computations of SKY increase smoothly to 10 and 15,000 respectively, which is negligible with respect to BASE with almost 200,000 similarity computations and consuming more than 140s time. Figure 5.6 shows that CBO has better performance than TBO by computing almost 25 similarity scores and consuming more than 25 seconds time.



(a) (b)

Figure 5.6: The results of experiments over 100k trajectories on WEBN. (a) Running time in each method vs different values of k; (b) Number of similarity score computations by each method vs k.

Moreover, we evaluate the performance of two variations of the ordered based method as CBO and TBO, by considering the number of visited query nodes during the searching process on CFN and WEBN (the results of experiments on OLN have mentioned earlier in Figure 5.4). Note that, the SkyLine method does not search for k-MsTraj over the nodes of the query, therefore its effectiveness does not depend on the number of visited query nodes. As shown in Figure 5.7 TBO outperforms CBO, significantly in terms of the number of visited query nodes. By increasing the k values to 64, the number of visited query nodes by CBO rise up quickly to 17 for the queries with length of 20, while this value for TBO is approximately two times less than the number of visited query nodes by CBO. This confirms the fact that ordering of query nodes by considering time information helps in raising the lower bound of trajectories at the beginning of searching process and makes Theorem 2 satisfied earlier. The difference between the performance of TBO and CBO, regarding the number of visited query nodes over 100k trajectories on WEBN, is almost negligible and is always about two times less than the length of the query (see Figure 5.7).



CFN-(a) WEBN-(b)

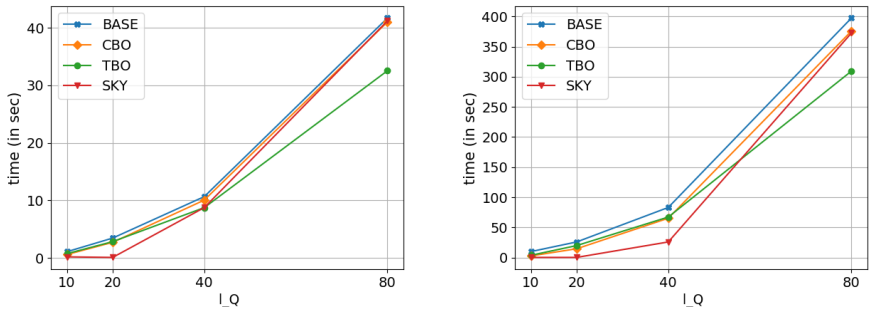
Figure 5.7: Evaluation of the performance of CBO and TBO w.r.t BASE in terms of the number of visited query nodes by varying the values k. (a) Experiments over 400k trajectories on CFN; (b) Experiments over 100k trajectories on WEBN.

Different Length of the Query Trajectory

Another concern about the k-MsTraj problem is how the algorithms behave by varying the length of the query trajectory. In this section, we fix k to 16 and compare the performance of the proposed methods regarding the length of the query trajectory as $10, 20, 40, 80$.

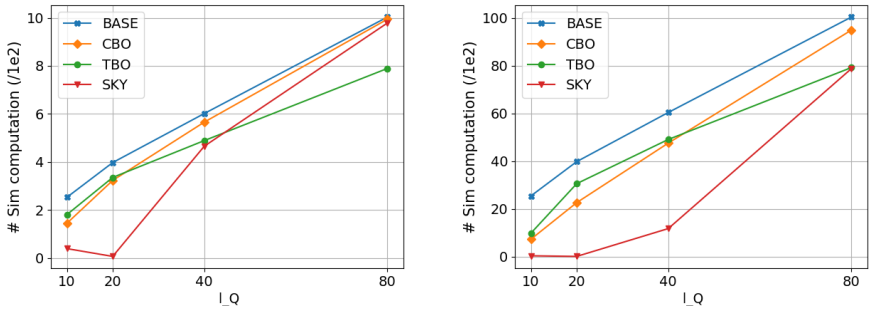
By considering two datasets over OLN containing 100k and 1M trajectories, as shown in Figures 5.8 and 5.9, we observe that the SkyLine method have better performance for a larger set of trajectories (i.e. 1M), in terms of the running time and the number of score computations by varying the length of the query. TBO outperforms other methods in terms of both running time and number fo score computations, for longer queries over both datasets containing 100k and 1M trajectories. Consecutively, we say that the performance of SkyLine on OLN depends on the number of trajectories in the dataset, since as shown in Figure 5.8 the values of time and number of score computations of SKY are getting less than other methods for the larger set of trajectories.

Figure 5.10 shows the results of experiments over 400k trajectories on CFN. SKY outperforms both CBO and TBO by computing fewer number of similarity scores and consecutively consuming less time for query processing. As we can see TBO in comparison with CBO in Figure 5.10, the values of time and the number of similarity computations in TBO is close to 12.5 and $5 \times 1e2$, respectively, for larger queries with length of 80, while these values for CBO are almost 17.5 and $6 \times 1e2$, respectively.



(a) (b)

Figure 5.8: Evaluation of the proposed method regarding the running time in comparison with BASE vs the number of query nodes on OLN (a) Experiments over 100k trajectories; (b) Experiments over 1M trajectories



(a) (b)

Figure 5.9: Evaluation of the proposed method regarding the number of score computations, in comparison with BASE vs the number of query nodes on OLN (a) Experiments over 100k trajectories; (b) Experiments over 1M trajectories

The experiments over 100k trajectories on WEBN shows the best performance of the SkyLine method regarding running time and the number of score computations. As the length of the query trajectory rises to 80, we have no significant rise for SKY in terms of running time and the number of score computations (see Figure 5.11)). Moreover, we can observe that CBO outperforms TBO significantly in terms of the running time and the number of score computations.

Regarding the number of visited query nodes, we can see in each dataset

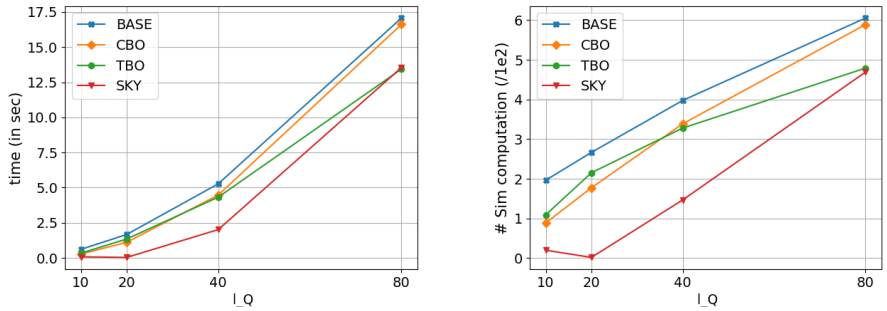


Figure 5.10: The results of experiments over 400k trajectories on CFN. (a) Running time in each method vs different number of query nodes; (b) Number of similarity score computations by each method vs different number of query nodes.

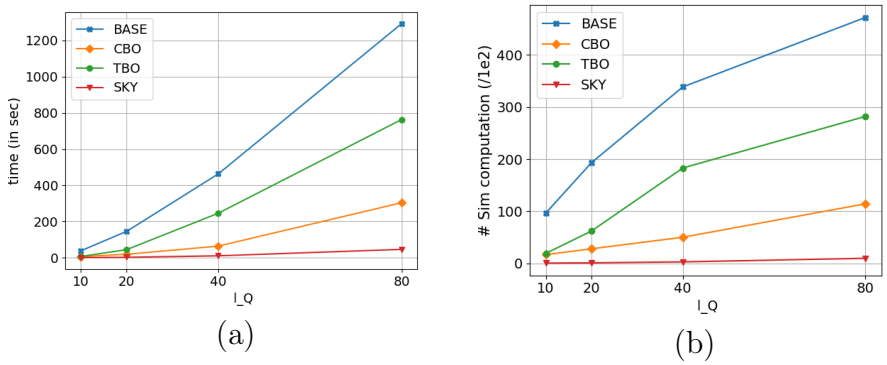
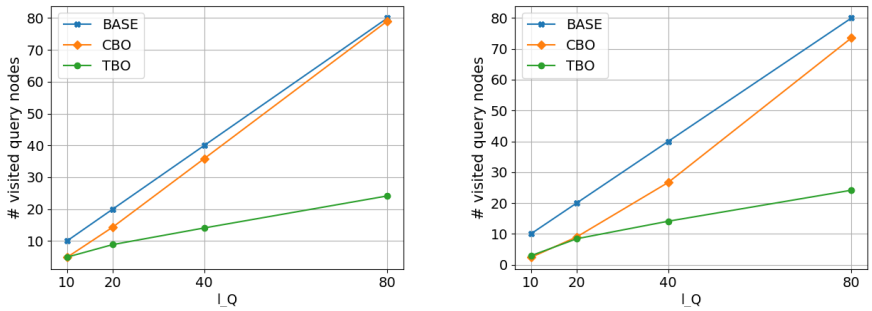


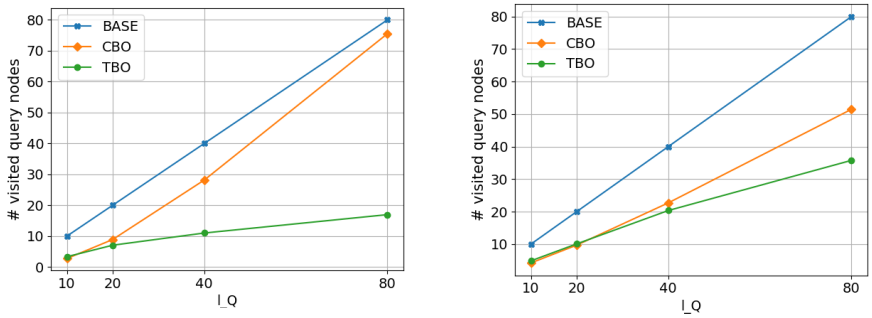
Figure 5.11: The results of experiments over 100k trajectories on WEBN. (a) Running time in each method vs different number of query nodes; (b) Number of similarity score computations by each method vs different number of query nodes.

(Figures 5.12–5.13), the the TBO method is more scalable with the length of the query in comparison with the CBO method, in terms of the number of visited query nodes. This is due to that, TBO discovers more trajectories at the beginning of the searching process. This helps to satisfy the requirements of the Theorem 2 earlier by visiting the fewer number of query nodes.



(a) (b)

Figure 5.12: Evaluation of the performance of CBO and TBO w.r.t BASE in terms of the number of visited query nodes by varying the length of the query on OLN. (a) Experiments over 100k trajectories; (b) Experiments over 1M trajectories.



CFN-(a) (WEBN-b)

Figure 5.13: Evaluation of the performance of CBO and TBO w.r.t BASE in terms of the number of visited query nodes by varying the length of the query. (a) Experiments over 400k trajectories on CFN; (b) Experiments over 100k trajectories on WEBN.

In summary, we conclude that all proposed methods in this chapter are effective in comparison with the BaseLine method, in terms of the running time, number of score computations and the number of visited query nodes by varying the k values and the length of the query. Moreover, we can drive that the SkyLine method is the best choice for computing k-MsTraj over the large datasets and the TBO method has better performance for processing longer queries for larger values of k in comparison with CBO.

5.2.3 Evaluation of the Approximate Algorithm

We evaluate the quality of the solution produced by the ApSky method w.r.t the exact solution provided by the SkyLine method. First, we discuss the effectiveness of ApSky in terms of the running time and number of similarity score computations in comparison with the SkyLine method in Figures 5.14–5.15. Then, we measure the accuracy of the produced solution by ApSky by means of the metrics described in Section 4.2.4 in Figures 5.16–5.21. We show the robustness of the methods evaluating each metric by varying values of k and the length of the query trajectory for each dataset.

Figure 5.14 shows the performance of ApSky in comparison with SkyLine, when the value of k changes from 1 to 64. As we can observe, the value of running time by ApSky rises up smoothly in each dataset in Figures 5.14 (a-d). This confirms that the number of trajectories processed by ApSky is much fewer than SkyLine, and consequently, as shown in Figures 5.14(e-h), the number of similarity score computations by SkyLine is almost twice and three times more than ApSky over 100k and 1M trajectories on OLN, respectively. We observe that this value is less on experiments over CFN and WEBN. This shows the better performance for SkyLine over these two networks. As we can see, for smaller values of k, there is a negligible difference between SkyLine and ApSky in terms of the running time and number of score computations. This is due to that the SkyLine method satisfies the requirement of the Theorem 2 on the outer layers of Pareto, for smaller values of k. These differences rise up, while the values of k getting larger. It means, the SkyLine method needs to search through more Pareto layers to make the Theorem 2 to be satisfied.

Similarly, Figure 5.15 shows the performance of ApSky in terms of running time and number of score computations, by varying the length of the query trajectory (i.e. the number of nodes of the query). As we can observe, the cost of ApSky is almost close to zero in comparison with SKY, over every four datasets. Regarding the performance of SKY and ApSky, these results are close with better performance for SKY. This is due that, ApSky continues to search through the Pareto layers until visits at least k trajectories. It means, ApSky always visits the minimum number of Pareto layers w.r.t the values of k. Therefore, if the performance of SKY is close to ApSky, shows that the SkyLine method could find the solution by satisfying the Theorem 2, within the first Pareto layers, which confirms the performance for the SkyLine method (see Figure 5.15).

In conclusion, we observe that the performance of ApSky is robust while the values of k and the length of the query rise up to 64 and 80, respectively.

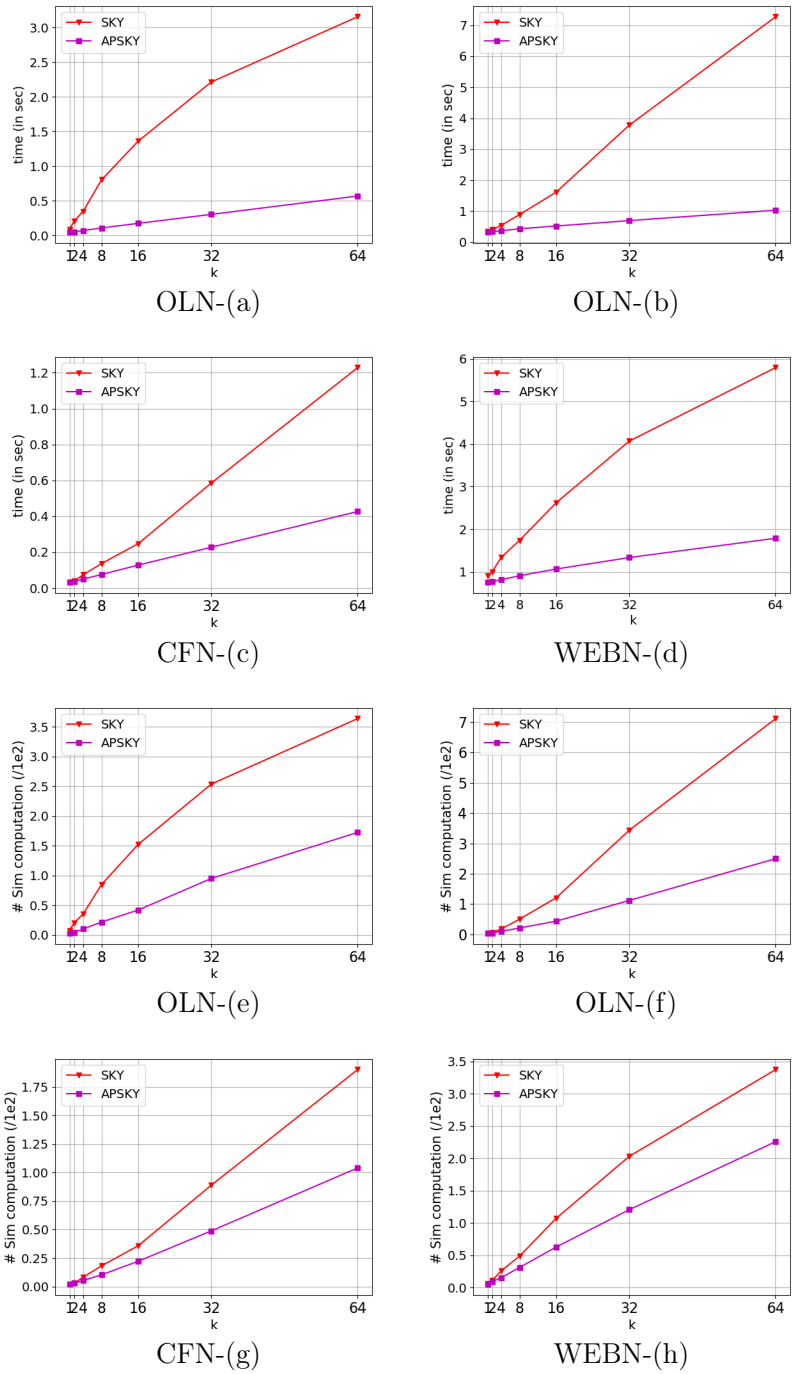
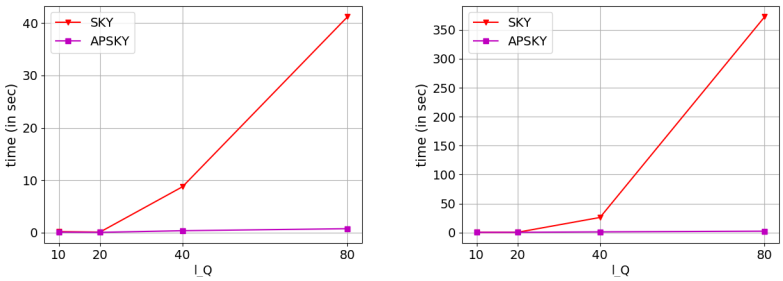
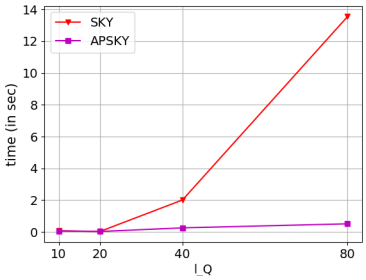


Figure 5.14: Evaluation of the approximate method (ApSky) in comparison with the exact method (SKY) by varying the values of k on 100k and 1M trajectories on OLN and 400k trajectories on CFN and 100k trajectories on WEBN. (a-d) Running time in each method; (e-h) Number of similarity score computations by each method.

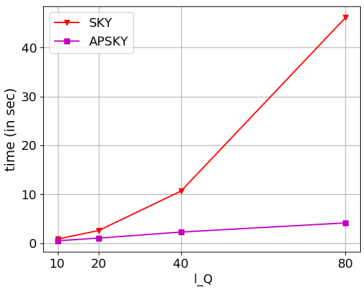


OLN-(a)

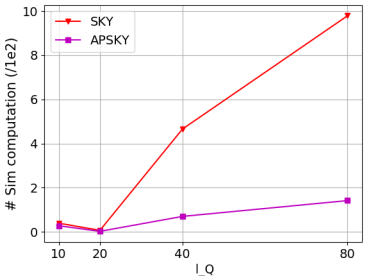
OLN-(b)



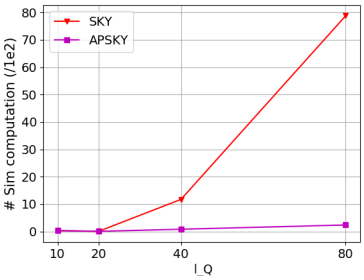
CFN-(c)



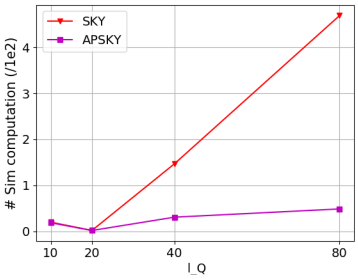
WEBN-(d)



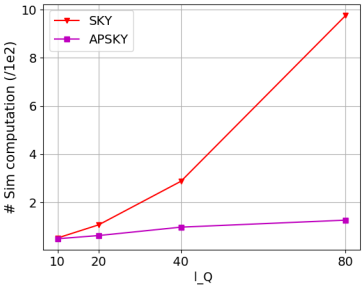
OLN-(e)



OLN-(f)



CFN-(g)



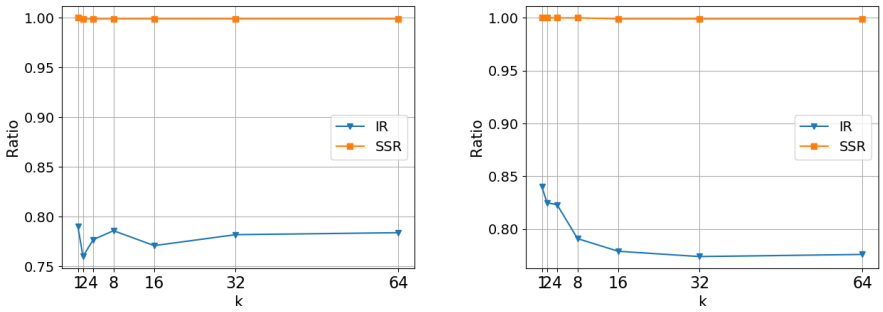
WEBN-(h)

Figure 5.15: Evaluation of the approximate method (ApSky) in comparison with the exact method (SKY) by varying the length of the query on 100k and 1M trajectories on OLN and 400k trajectories on CFN and 100k trajectories on WEBN. (a-d) Running time in each method; (e-h) Number of similarity score computations by each method.

Precision Ratios: IR and SSR

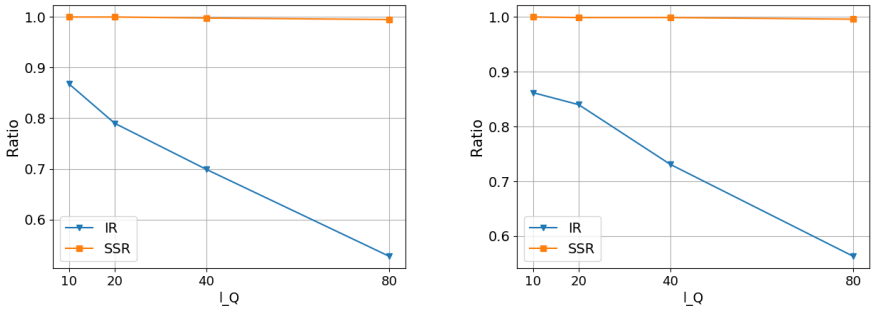
Regarding the quality of the provided solution by ApSky, the results in Figures 5.16–5.21 shows the *intersection ratio*(IR) and similarity score ratio(SSR) ratios for each dataset. Given two output sets γ_1 and γ_2 containing top$-k$ trajectories, e.g. the exact and approximated solutions, respectively, we have: $\text{SSR}(\gamma_1, \gamma_2) = \frac{\sum_{T \in \gamma_1} Sim(Q, T, t)}{\sum_{S \in \gamma_2} Sim(Q, S, t)}$ and $\text{IR}(\gamma_1, \gamma_2) = \frac{|\gamma_1 \cap \gamma_2|}{k}$.

As shown in Figure 5.16 the IR values are robust by varying the values of k almost around 0.80 for both datasets containing 100k and 1M trajectories on OLN. Regarding the different values of the length of the query, shown in Figure 5.17, we can observe that the IR values changing from almost 0.90 to 0.60 while the length of the query rises up to 80 over both datasets on OLN.



100k 1M

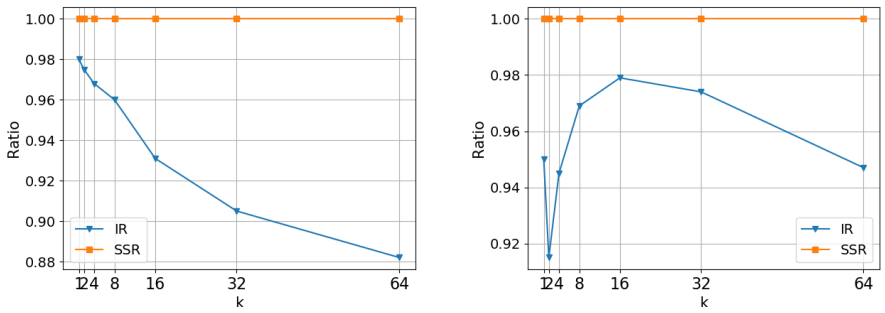
Figure 5.16: IR and SSR ratios on OLN vs k



100k 1M

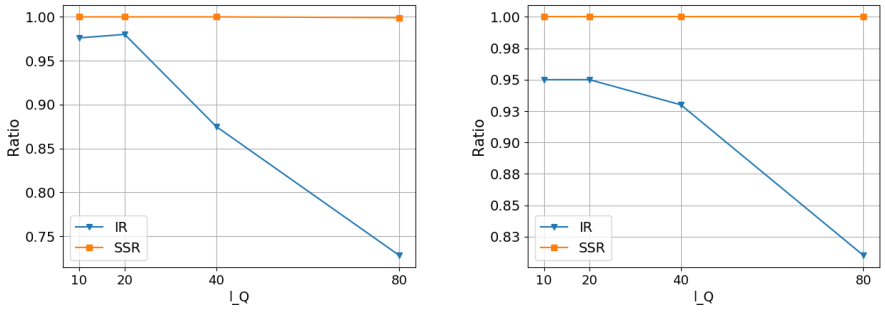
Figure 5.17: IR and SSR ratios on OLN vs l_q

On the other hand, the values of the IR ratio over 400k trajectories on CFN go down from 0.98 to 0.89, as the value of k rises up to 64. Similarly, considering a larger dataset on CFN containing 4M trajectories, the IR values change within the range of 0.91 to 0.98. This value reaches to 0.98 when k is equal to 16 (see Figure 5.18). Moreover, we can see in Figure 5.19 that IR values are close to one for shorter query trajectories in both datasets 400k, 4M and getting close to less than 0.75 and 0.83, over 400k and 4M trajectories, respectively.



400k 4M

Figure 5.18: IR and SSR ratios on CFN vs k



400k 4M

Figure 5.19: IR and SSR ratios on CFN vs l_q

The ApSky has the best performance in terms of IR ratio on WEBN, since as Figure 5.20 shows, the IR values started from 0.99 reaching to 0.98 and then for larger values of k returned closing to 0.99. In addition, in Figure 5.21, we can see the good quality of ApSky for different values of the

length of the query. The IR started from 0.99 getting close to 0.96 for the queries with a length of 40 and then rises up to more than 0.88 while the length of queries reaching to 80. This is due that, WEBN is a dense network, with the diameter 11, and a trajectory with maximum similarity score moves close to a more number of query nodes, in comparison with other networks with a larger diameter, and consequently, it is easier for ApSky to find such trajectories.

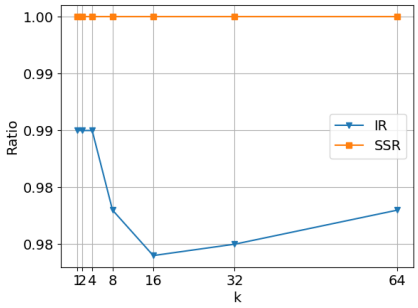


Figure 5.20: IR and SSR ratios over 100k trajectories on Web-network vs k

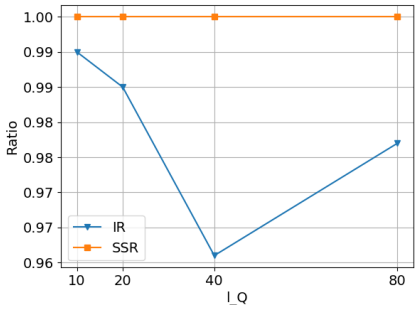


Figure 5.21: IR and SSR ratios over 100k trajectories on Web-network vs l_q

Indeed, we observe that the SSR ratio is always very close to 1, for each dataset, regarding different values of k and the length of the query. This confirms the high performance of the approximated solution for the k-MsTraj problem.

5.2.4 Brinkhof Dataset

To evaluate the scalability of the results of our experiments, we conduct some other experiments over 100k trajectories which are generated randomly by Brinkhof generator on the Oldenburg network. The presented results correspond to the average of 100 Brinkhof generated queries executed in each case.

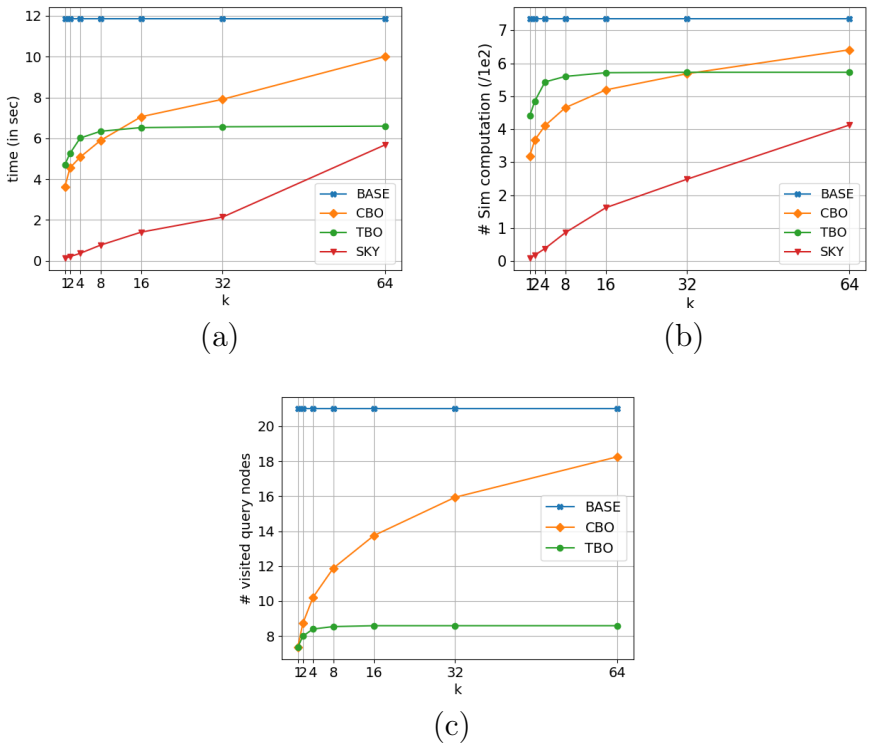


Figure 5.22: Evaluation of exact methods over 100k trajectories is generated by Brinkhof generator on OLN network vs k. (a) Running time by each method in comparison with BASE; (b) The number of similarity score computations by each method; (c) The comparison of the number of visited query nodes by each method during the searching process.

Exact Methods

As shown in Figure 5.22, SKY notably outperforms BASE and other methods CBO and TBO. As the value of k goes up to 64, the running time and the number of score computations of CBO rise quickly to more than 10

and 6, respectively. In addition, the running time and the number of score computations by the TBO method are robust for different values of k and are around 6. While SKY rises the cost smoothly with running time almost 5 and the number of score computation about 4.

Note that the running time and the number of score computations by BASE are fixed for different values of k with values the of 12 and 7, respectively. This is due that the searching process in the BaseLine algorithm does not depend on the k values.

Indeed, regarding the number of visited nodes by each method, we have a comparison between CBO and TBO in Figure 5.22 to show that TBO outperforms CBO. In comparison with TBO, when the query is ordered without time consideration (CBO), more query nodes are visited and consequently it requires processing more trajectories to figure out the lower bound by using the Algorithm 3. Therefore, more score computations and more time are needed, particularly, for larger values of k, as confirmed in Figure 5.22.

Approximated Method

As shown in Figure 5.23, the approximation method ApSky outperforms the exact method SkyLine in terms of the running time and the number of score computations, significantly. The IR and SSR ratios plotted in Figure 5.23 show the quality of the produced solution by ApSky in comparison with the exact solution. We observe that as k goes up to 64, the IR value starts from 0.85 going down to more than 0.70 when $k = 8$ and then rises up to 0.80 for larger values of k.

5.3 Conclusion

In this chapter, we proposed and investigated the exact (i.e. CBO, TBO, SkyLine) and approximate (i.e. ApSky) techniques based on the prune-and-refine method, treating the k-MsTraj problem. We aimed at reducing the number of similarity scores that need to be computed, consequently, reducing the number of shortest path distance computations and in general, the time needed to process the query. We proposed the Ordered-Based technique that aims at visiting less number of query nodes during the searching process, and we discussed two variants of this method: CBO, which processes the nodes of the query w.r.t the number of trajectories passing through each node; and TBO, which processes the nodes of the query, w.r.t the number fo trajectories passing through each node within the corresponding time intervals. In addition, we presented the SkyLine method based on the dominance

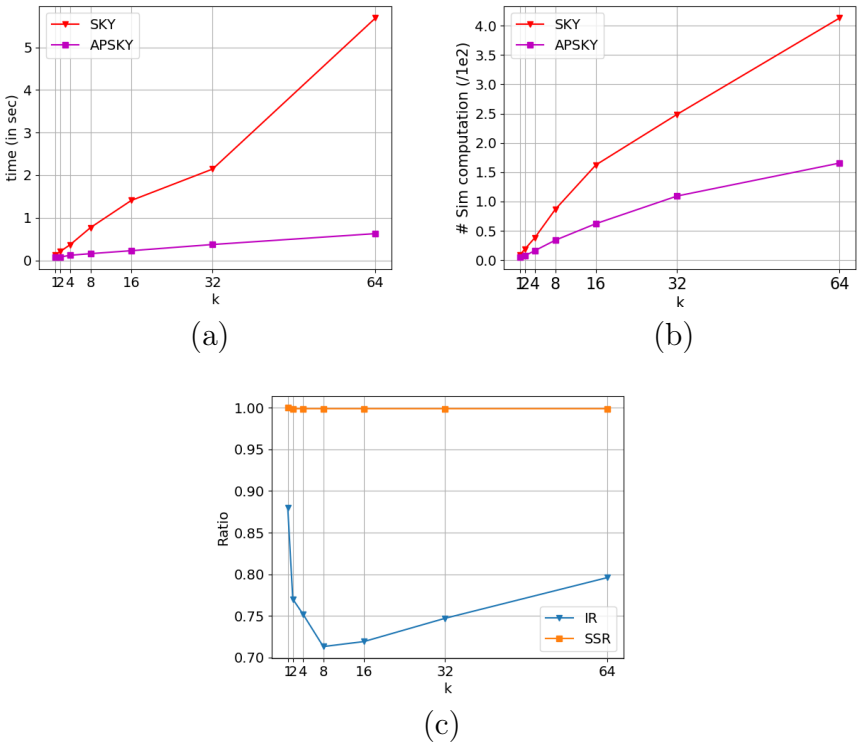


Figure 5.23: Evaluation of approximate method over 100k trajectories is generated by Brinkhof generator on OLN network vs k. (a) Running time in ApSky in comparison with SKY ; (b) The number of similarity score computations; (c) The IR and SSR ratios (i.e. the quality of the produced solution by ApSky)

relationships between trajectories, aiming at reducing the number of trajectories that must be processed. In all methods, we provided a pair of bounds (i.e. lower and upper bound) to manage the pruning of the search space efficiently. In contrast to state-of-the-art works which process the query in the temporal and spatial domain separately [79, 81, 41], we treated the k-MsTraj query processing in a single searching process, in both proposed methods. Moreover, by making use of the SkyLine method, we provided a fast and accurate approximation technique (i.e. ApSky).

Finally, the performance of the proposed methods was verified through extensive experiments based on synthetic trajectories generated on three real-world networks: Oldenburg, California and Web network. We observed that all proposed methods are effective in comparison with the baseline, in terms of the query time, number of score computations and the number of visited

query nodes by varying the k values and the length of the query. We conclude that the SkyLine method is the best choice for computing the k-MsTraj over the large datasets and the TBO method has better performance for processing longer queries for larger values of k in comparison with the CBO.

Concerning the approximation method, we showed the performance of ApSky, in terms of the time needed to process the query and the number of score computations, which always outperforms the SkyLine. In addition, we measured the accuracy of the produced results in terms of two proposed precision metrics. We demonstrated the precision of the method over each dataset, which shows that ApSky can make fast and reasonably accurate results.

In comparison to the approximation algorithms provided in Chapter 4 (i.e. SHQ and SHQT), the results of experiments over the same dataset WEBN show that ApSky works well for larger datasets and has the best performance for small values of k, while more accurate results can be obtained by SHQ and SHQT when considering long trajectories for large values of k.

It is worth to mention that, in many domains such as urban planning, traffic management, intelligent transportation systems, nowcasting, and emergency management, faced with the huge size of trajectory set, specifying the *purpose* of the movements, in order to have a better comprehension of the system in each domain is a big challenge, which can benefit of the results of this chapter.

Chapter 6

Next Movement Prediction of Tourists

In the last years, increasing availability of user-generated content has driven the development of new services for tourism. The new services target different tasks ranging from, for example, prediction of next places to visit [60, 9, 60], intelligent planning [2, 15, 14], booking, etc. These systems are used within location-based social networks and targeted mobile applications to help tourists in achieving a better experience in their activities.

In this chapter, we study the next-PoI prediction problem (next-PoI), which aims at identifying the PoI that a tourist will visit in the future with highest probability. Usually, this problem is solved by using machine learning-based techniques. The overall efficiency of a machine learning-based technique for next-PoI prediction depends on the effectiveness of the proposed feature set. In particular, given a trajectory dataset, any machine learning model needs to extract a set of useful features for prediction, which is a challenging task. In this project, we introduce a new graph-based method targeting the next-PoI problem. The objective of the next-PoI is to reflect the similar behavior of the past tourists, which is really useful in many applications, to predict the next movements of a tourist.

To tackle this problem, we exploit the structural and temporal information available in the past trajectories of tourists, i.e., the sequences of PoIs visited by tourists in a city, to predict the next one, i.e., the PoI that will more likely be visited. We introduce the *trajectory graph*, a graph modeling the behavior of a given set of tourists in a city. Using this graph, we identify the trajectory which is the most similar to a given one in order to predict the next-PoI.

We propose a comprehensive evaluation of our proposed method and

of state-of-the-art competitors (based on machine learning) on three public datasets of movements in Pisa, Rome, Florence. The performance of the methods, evaluated in terms of Success@k% (i.e., the percentage of times that the correct answer is in the top-k ranked PoIs), showing that our method achieves the best performance, outperforming the competitors.

This chapter is structured as follows: Section 6.1 provides a formal definition of the problem. In Section 6.2 we present the needed concepts to explain the prediction model that is discussed in detail in Section 6.3. Finally, in Section 6.4 we report the experiments. We illustrate the effectiveness of our method compared to state-of-the-art techniques. As a result, we show that our method can predict next-PoI for tourists with high accuracy.

6.1 Problem Statement

Let $P = \{p_1, p_2, \cdots, p_n\}$ be a set of Points of Interests(PoI) of a city. Let $U = \{u_1, u_2, \cdots, u_m\}$ be a set of users. We consider the movements of the user u while visiting the subset of PoIs, as a sequence of pairs $(p_i, t_i = [s_i, e_i])$ where s_i and e_i are the time instances u enters to and exits from p_i, respectively. We define a temporal order of these pairs as the *trajectory* of the user u, following Definition 1, as $T = \langle (p_1, [s_1, e_1]), (p_2, [s_2, e_2]) \cdots, (p_N, [s_N, e_N]) \rangle$.

The set of all trajectories $\mathcal{T}$ contains the trajectories T of all the users $u \in U$.

Problem Statement Given a set of trajectories $\mathcal{T}$ and a user u, let Q be the trajectory of u in $\mathcal{T}$, with $Q = \langle (p_1, [s_1, e_1]), (p_2, [s_2, e_2]) \cdots, (p_N, [s_N, e_N]) \rangle$. We aim at predicting the PoI in P that the user u will visit after the PoI p_N.

6.1.1 Sketch of the Solution

Our basic idea is to use the relation between the trajectory of the tourists to predict their next movement. Given a set of trajectories $\mathcal{T}$ and a query trajectory Q, we solve the problem by performing two major steps, which are illustrated in Algorithm 5. We first construct a graph of tourist trajectories on the city, called *trajectory graph* (alternatively we can use graph), for a given dataset of trajectories. In a trajectory graph, there is an undirected edge (p_i, p_j) if there exists a trajectory passing through (p_i, p_j). In the second step, we find Most Similar Trajectory (MsTraj), that is, k-MsTraj (when $k = 1$) described in Section 3, to a given query trajectory based on the similarity function. Computing MsTraj for a given trajectory of a tourist is a fundamental task to solve our problem. This is because the current PoI

visited by MsTraj on the graph could be a PoI that the given tourist will visit in the future. We present the needed concepts to explain the solution in Section 6.2 and then we discuss the method in detail in Section 6.3.

Algorithm 5: Two major steps for PoI prediction

Input: $\mathcal{T}, Q$
Result: next movement of Q

begin
  step1: Build the trajectory graph G w.r.t $\mathcal{T}$ and Q
  step2: Find MsTraj in $\mathcal{T}$ w.r.t Q over G
  Report the current location of MsTraj as the next movement of Q

6.2 Using Trajectory Similarity on Graphs

In this section, we present some preliminary concepts needed to propose the solution for the PoI prediction problem.

Since the motion of tourists is restricted by the PoIs in a city, we create a graph whose nodes are the PoIs they visited and edges are placed depending on the movements in the trajectories, as described next.

Definition 8 (Trajectory Graph) *Given a set of trajectories $\mathcal{T}$ visiting a set of PoIs P, $G = (V, E)$ is an undirected graph whose node set V is P and whose edge set E is such that for any two nodes $p_i, p_j \in V$ there is an undirected edge $(p_i, p_j) \in E$ if there exists at least one trajectory in $\mathcal{T}$ that traversed p_i and p_j, consecutively.*

We further use the similarity measure described in Section 3.1.2 to compute the similarity between two trajectories. For a given trajectory, we define the Most Similar Trajectory (MsTraj) as the trajectory with the maximum similarity according to Definition 3.

Given a query trajectory Q, by considering $Sim(Q, T_j, t)$ as the similarity between the query Q and the trajectories $T_j \in \mathcal{T}$ during the time interval $t = [a, b]$, we define the *next-PoI* of Q as follows.

Definition 9 (next-PoI) *Given a set of trajectories $\mathcal{T}$ over the graph $G(V, E)$, a trajectory $Q = \langle (p_1, t_1), (p_2, t_2) \cdots, (p_N, t_N) \rangle$ of a tourist, and a query time interval $t = [a, b]$, the next-PoI of Q is a node $p_i \in V$ which maximizes:*

$$\rho_{Ni} \times \tau_{jie} \times Sim(Q, T_j, t = [a, b]), \tag{6.1}$$

where:

$$\rho_{Ni} = \begin{cases} 1 & \textit{if } (p_N, p_i) \in E \\ 0 & \textit{otherwise} \end{cases}$$

$$\tau_{jie} = \begin{cases} 1 & \textit{if } T_j(b) \textit{ defined and } T_j(b) = p_i \\ 0 & \textit{otherwise} \end{cases}$$

The next-PoI problem, based on the above definitions, is then the following one: given a set of trajectories $\mathcal{T}$ and a query trajectory Q, find the next-PoI of Q over the trajectory graph G. Note that G is the trajectory graph derived from $\mathcal{T}$ and Q.

6.3 Proposed Method

Given the set $\mathcal{T}$ and a query trajectory Q, we first build the trajectory graph G with respect to $\mathcal{T}$. Note that this can be computed in an offline manner as follows. In order to build G, we simply scan all trajectories in $\mathcal{T}$ and update G during the process. Initially, G is an empty graph. For all trajectories $T \in \mathcal{T}$ and for each consecutive pair $(p_i, [s_i, e_i]), (p_j, [s_j, e_j]) \in T$, if the edge (p_i, p_j) does not exist in G, we add it to G. The construction of G is completed once all trajectories are considered.

When processing the given query trajectory Q, we update G by scanning Q. Now, for finding next-PoI, we go through the second step of the Algorithm 5 over G, finding the trajectory most similar to Q in $\mathcal{T}$ as follows.

Finding MsTraj Given a set of trajectories $\mathcal{T}$ defined over a graph G, a query trajectory Q and a time interval $t = [a, b]$, we aim at finding MsTraj. To this end, we simply compute the similarity score of each trajectory $T \in \mathcal{T}$ with respect to the query Q within t based on the similarity function introduced in Definition 3. Obviously, in the present of $t = [a, b]$, we should only consider those trajectories which are defined for each time instance $i \in t$. We maintain all trajectories in an ordered heap H, according to their similarity scores. Thereafter, we report the trajectory on the top of the heap H (i.e. with maximum similarity score) as MsTraj.

Finding the next-PoI Given a trajectory $Q = \langle (p_1, t_1), (p_2, t_2) \cdots, (p_N, t_N) \rangle$ and a time interval $t = [a, b]$. We assume the neighbor nodes of p_N in the trajectory graph G, denoted by $NB(p_N)$, as the candidate next-PoI of Q.

All such nodes $p_i \in NB(p_N)$ are indeed such that the value $\rho_{Ni} = 1$ in Equation 6.1. Therefore, for the query Q and time interval $t = [a, b]$, if there exists a trajectory $T \in \mathcal{T}$ such that $T(b) \in NB(p_N)$ and satisfies Equation 6.1 in Definition 9, then we consider $T(b)$ as the next-PoI of Q, where recall that $T(i)$ indicates the unique node $p \in V$ such that there exists a pair $(p, t) \in T$ with $i \in t$. On the other hand, let T be MsTraj, if $T(b) \in NB(p_N)$ then T obviously satisfies the Equation 6.1, since has the maximum similarity score, so $T(b)$ is the next-PoI of Q. Therefor, for a trajectory T, $T(b)$ is the next-PoI of Q, if $T(b) \in NB(p_N)$ and T is MsTraj w.r.t Q.

Based on this idea, we address the next-PoI problem, for given query $Q = \langle (p_1, [s_1, e_1]), (p_2, [s_2, e_2]) \cdots, (p_N, [s_N, e_N]) \rangle$. We consider $[s_N, e_N]$ of the current pair $(p_N, [s_N, e_N]) \in Q$ as the query time interval $t = [a, b]$, such that $a = s_N$ and $b = e_N$. Let H be the heap associated to the set of trajectories in $\mathcal{T}$ based on their similarity scores w.r.t Q within t. In searching process, we consider only the trajectory on the top of the heap H and investigate whether the top trajectory satisfies the Equation 6.1. We continue to visit top-ranked trajectory in H finding the trajectory T maximizing the Equation 6.1. Thereafter, $T(e_N)$ will be reported as the next-PoI for the query Q.

We will evaluate the quality of our proposed model by conducting the comprehensive experiments in Section 4.2.

6.4 Experimental Evaluation

In this section we report the experiments to evaluate our proposed method. We describe in Section 6.4.1 the three real-world datasets employed for the experiments, and the next-PoI prediction framework in Section 6.4.2. We compare our solution with the state-of-the-art methods described in Section 6.4.3, adopting the methodology discussed in Section 6.4.4. In particular, we pose four questions that are experimentally analyzed in Sections 6.4.5–6.4.8 to show the effectiveness of our methods.

6.4.1 Datasets

To evaluate our method we use three different datasets provided in [60] containing tourist movements covering three Italian cities Pisa, Rome, Florence. In these datasets, the set of PoIs in each city is aggregated from Wikipedia.

For making tourist trajectories, the geo-tagged photos from Flickr are collected. The photos are mapped to the set of PoIs aggregated from Wikipedia. Note that, since users may have taken more than one photo assigned to the same PoI, the timestamps of the first and the last photo are considered as the starting and ending time of their visits at the PoI. For each Flickr user, the sequence of these PoIs makes its movement pattern in each city. Each sequence of PoIs visited by each user is split into the set of daily journeys, which builds the trajectory set. Apart the user ID, the format is a quadruple with the PoI ID, the number of pictures taken there, and the timestamp for the first picture and that for the latter (timestamps are in ms with reference to Jan. 1, 1970). For example, this is an example of a line for a raw dataset, where two quadruples are assigned to the same line:

```
25615082@N03 <tab> P93;2;1307114479;1307114577 <tab> P67;1;1307115649;1307115649
```

The above line means that user `25615082@N03` took two pictures in PoI `P93` and then one picture in PoI `P67`. Note that each line corresponds to a user ID, but more lines can be associated with the same user ID, with one or more quadruples in each line separated by a tab.

In our experiments, we consider the time duration between the starting time assigned to two consecutive visited PoIs p_i and p_{i+1} by a user, as the time duration the user spent in p_i. In the above example, the first PoI `P93` has the time interval $[1307114479, 1307115648]$. The last PoI in each line has the same timestamp for beginning and end as $[1307115649, 1307115649]$ in our example. Note that this is not a limitation, as the last PoI is the one to be predicted in our experiments. In this way each trajectory is represented in the form of the Definition 1.

Table 6.1 shows the properties of the datasets we use. The column #PoI shows the number of PoIs visited by trajectories in each dataset. Moreover, we report the number of trajectories we consider in our experiments which are the trajectories visit at least two PoIs.

Dataset	#PoI	#Trajectories ≥ 2
Pisa	110	992
Florence	888	5984
Rome	490	12565

Table 6.1: Properties of three datasets

The distribution of the length of trajectories for three cities are plotted in Figure 6.1

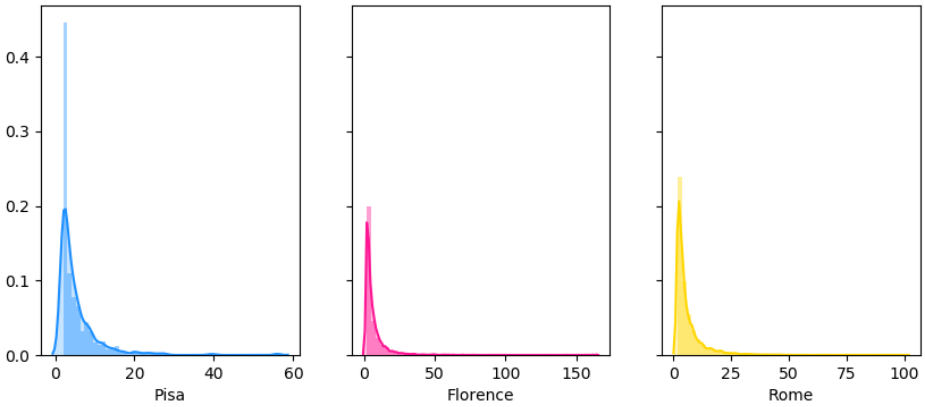


Figure 6.1: The distribution of the trajectories length in each dataset

6.4.2 Experimental Environment

We perform the evaluations using a machine with Intel(R) Xeon(R) CPU E5-2620 v3 at 2.40GHz, 24 virtual cores, 128 Gb RAM, running Ubuntu Linux version 4.4.0-22-generic. We implement all algorithms in `Python3`.

6.4.3 Methods

We compare our method with a probability baseline PROB and LEARNEXT [60] described below. Although there are two important state-of-the-art techniques WHERENEXT [59] and Random Walk [53], we do not include them in our experimental study, since LEARNEXT outperforms them. We evaluate our method using the same datasets and training/test sets employed in [60].

PROB A pure model named "PROB" in [60], uses the trajectories in dataset to build a weighted-directed graph with PoIs as the node set. There is a directed edge from source PoI p_i to the destination PoI p_i, if there exists at least one trajectory in dataset that traverse from p_i to p_j. Each edge is weighted with the number of transaction from source to destination, representing the *transition probability* of each edge. Given p, the method PROB returns the out-neighbor of p with highest probability. Intuitively, this model suggests the most visited PoI from the source p as the next-PoI.

LEARNEXT LEARNEXT is a next-PoI predictor that learns tourists' behavior from their common patterns of movements. Authors apply machine learning techniques to tackle the problem of predicting the "next" touristic attraction a user will visit on the basis of her visit history (i.e., the prediction is done accordingly to what the user has already visited in the touristic attraction). The problem is modeled as an instance of learning to rank which exploits a feature space composed of 68 features capturing both the touristic behavior and the peculiar characteristics of candidate PoIs. The models are trained using GBRT [100] and Ranking SVM [43] as learning methods. They are tested on three collections of touristic trajectories corresponding to popular Italian touristic areas, in particular, data from photos taken in *Pisa*, *Florence*, and *Rome*, which are also used in this study.

6.4.4 Evaluation strategy

The evaluation of our proposed solution is aimed at answering the following research questions:

Q1: Is our proposed solution effective for predicting the next-PoI?

Q2: What is the effectiveness of the proposed solution regarding the varying the scale of the temporal aspect of trajectories (i.e. year, month, day, hour)

Q3: How do the following parameters affect the prediction?

- The popularity of the PoIs (i.e. the number of trajectories passing through the PoIs)
- The degree and centrality of PoIs on the graph
- The geographic (Euclidean) distance between PoIs
- The length of the time intervals are assigned to the query PoIs

Q4: What is the robustness of the proposed method regarding the adopted similarity functions?

To answer these questions, we follow the same evaluation strategy adopted in [60] over the three aforementioned datasets, which is a standard training/test evaluation strategy. For each city, we consider 80% of trajectories as a training set S and 20% of trajectories as the test set S'. The effectiveness of the methods are assessed by means of Success@k (i.e., the percentage of times that the correct answer is in the top-k ranked PoIs) [66, 60]. Specifically, we will use $k = 1$ in our evaluations, which is the topmost PoI.

Algorithm 6: Evaluation Block

Input: Training set S
Input: $Q = \langle (p_1, t_1), (p_2, t_2) \cdots, (p_{N-1}, t_{N-1}), (p_N, t_N) \rangle \in$ test set S'
Result: next-PoI

62 Build G by trajectories in S
63 Make $Q.tail$ and $Q.head$
64 Update G by $Q.head$
65 $t = t_{N-1}$
66 Heap H
67 **for** *each trajectory* T *in* S **do**
68 H.add(T) w.r.t $Sim(Q, T, t)$
69 **while** $heap - size(H) < k$ **do**
70 MsTraj $\leftarrow pop(H)$
71 $p \leftarrow$ last position of MsTraj within t_{N-1}
72 **if** $p \in NB(p_{N-1})$ **then**
73 report p as the next-PoI
74 break

As illustrated in Algorithm 6, to evaluate the effectiveness of the proposed method, we first build the trajectory graph G, by using the trajectories in the training set S (Line 62). Then, for building the query set, we make use of the following process: We divide each trajectory in the test set S' into two parts: Head and Tail (Line 63). Let $Q = \langle (p_1, t_1), (p_2, t_2) \cdots, (p_N, t_N) \rangle$ be a query trajectory in S'. The tail of Q is the last pair of trajectory, denoted as $Q.tail = \langle (p_N, t_N) \rangle$, and the first $N-1$ pairs of Q makes the head which is $Q.head = \langle (p_1, t_1), (p_2, t_2) \cdots, (p_{N-1}, t_{N-1}) \rangle$.

For each trajectory like Q in the test set S', we aim at using the $Q.head$ and predicting the PoI p_N of $Q.tail$ as the next movement of Q. To this end, we update the graph G by making use of the $Q.head$ in Line 64. Then, we use the graph G as the underlying graph for trajectories to predict the $Q.tail = \langle (p_N, t_N) \rangle$. Based on Definition 9, the next movement of $Q.head$ would be a neighbor PoI of p_{N-1} on G. Let $NB(p_{N-1})$ denotes the neighbor PoIs of p_{N-1} on G. We aim at using the $Q.head$ to choose a right PoI $\in NB(p_{N-1})$.

We consider the last pair $\langle (p_{N-1}, t_{N-1}) \rangle$ of $Q.head$ as the current location of Q. Through the lines 65-68, by considering t_{N-1} of the current location of Q as the query time interval t in Definition 3, we compute the similarity score between each trajectory in training set and Q within the time inter-

val $t = t_{N-1}$. Obviously, we first restrict Q within t, then we compute the similarity scores. Moreover, we make an ordered heap of H of trajectory ids regarding their similarity scores. The trajectory with maximum score (MsTraj) is on the top of H. We consider the last PoI p of MsTraj within t_{N-1} as the next-PoI, if $p \in NB(p_{N-1})$. Otherwise, we continue to search over the trajectories in H (the *while* loop). In each round, we pick a trajectory with maximum similarity. We stop searching, if we find a MsTraj with a last position in $NB(p_{N-1})$ within t_{N-1} (lines 70-74).

Thus, the main task of the evaluation method is to measure how many times our model is able to choose the right PoI in terms of Success@1, by following the aforementioned strategy.

6.4.5 Question Q1: Effectiveness

In this part, we investigate whether the proposed model is an effective model for predicting the next-PoI of a given tourist trajectory. In the first experiments, we measure metric Success@1. Results are provided for our proposed method MsTraj along with the two methods (Prob and LearNext). Table 6.2 shows the results of the experiments, where we show that our method outperforms the competitors (the results of the proposed method is highlighted). As we can observe, MsTraj provides almost twice more accurate results than LearNext in terms of Success@1 in each city. While MsTraj provides almost six times more accurate results than Prob for Pisa and Rome, and almost ten times more for Florence, confirming the effectiveness of our method.

Dataset	Predictor	Success@1 %
Pisa	Prob	15.57
	LearNext	40.70
	MsTraj	**67.33**
Rome	Prob	12.59
	LearNext	30.95
	MsTraj	**77.96**
Florence	Prob	4.96
	LearNext	37.56
	MsTraj	**53.57**

Table 6.2: Effectiveness in terms of Success@1 of the proposed method (MsTraj) along with the competitors

Dataset	Query Time Unit in millisecond	Success@1 %
Pisa	yyyy-MM-dd HH:mm:ss	67.33
	MM-dd HH:mm:ss	65.32
	dd HH:mm:ss	**68.84**
	HH:mm:ss	65.32
Rome	yyyy-MM-dd HH:mm:ss	77.96
	MM-dd HH:mm:ss	**79.48**
	dd HH:mm:ss	78.72
	HH:mm:ss	79.24
Florence	yyyy-MM-dd HH:mm:ss	53.57
	MM-dd HH:mm:ss	54.24
	dd HH:mm:ss	54.83
	HH:mm:ss	**55.08**

Table 6.3: Effectiveness of the proposed method by varying the unit of time intervals assigned to PoIs (i.e. yyyy-MM-dd HH:mm:ss)

6.4.6 Question Q2: Varying the scale of the time query

To have a comprehensive evaluation of the proposed method, we conduct four experiments by varying the time unit of the intervals assigned to the query PoIs in the dataset. Let $Q = \langle (p_1, t_1), (p_2, t_2) \cdots, (p_{N-1}, t_{N-1}), (p_N, t_N) \rangle$ is a trajectory in the test set. As we mentioned earlier, we are interested in the pair $\langle (p_{N-1}, t_{N-1}) \rangle$ of $Q.head$. Thus, we compute the similarity between $Q.head$ within the time interval $t_{N-1} = [s_{N-1}, e_{N-1}]$. Note that the time instances s_{N-1}, e_{N-1} in the three datasets are milliseconds, which can be converted in the date format of yyyy-MM-dd HH:mm:ss. For example, assume we are interested in studying the trajectory of tourists in a specific event which is happened in a specific year in a city. In this case, it is important to know the exact date and time that each user visits each PoI. In this case, we capture the next-PoIs by considering the time instances in milliseconds for query time interval t_{N-1} in 67.33%, 53.57% and 77.96% in three datasets Pisa, Florence, and Rome, respectively. On the other hand, someone may be interested in studying the behavior of all tourists who visit a city only in one specific month like August. Thus, we drop the year information on trajectories. Thus, as the next experiment, we forced the time instances to the same year. In this case, by considering the times in milliseconds in the format of MM-dd HH:mm:ss, we predict the next PoIs in 65.32%, 54.24% and 79.48% for Pisa, Florence and Rome, respectively. Similarly, we repeat the experiments by considering time instances in milliseconds of the formats dd HH:mm:ss and HH:mm:ss. The results are shown in Table 6.3.

We highlighted in bold the best performance. We do not consider PROB and LEARNEXT in these experiments, since both of these methods predict the PoIs without considering time aspect of trajectories.

6.4.7 Question Q3: Different Parameters

The main task of this part is to study the performance of the proposed model w.r.t the parameters previously defined. For a given query Q, since, we use the current PoI of $Q.head$ to predict $Q.tail$, the evaluation strategy must contain the properties of PoIs of $Q.head$ and $Q.tail$. We will discuss it in detail in the following.

Dataset	Predictor	Success@1 %			
		Popularity		Geographically Closeness	
		F	R	GC	GF
Pisa	PROB	27.27	1.12	29.29	2.02
	MSTRAJ	88.18	44.94	88.88	48.48
Rome	PROB	21.69	8.93	23.44	1.51
	MSTRAJ	94.68	71.55	90.03	65.89
Florence	PROB	11.95	2.45	8.55	0.33
	MSTRAJ	86.45	44.77	67.28	39.69

Table 6.4: Effectiveness of the proposed method on predicting either Frequently (F) or Rarely (R) visited PoIs and either geographically Close (GC) or Far (GF) visited PoIs

The popularity of the PoIs In this part, we aim at evaluating the effectiveness of the proposed method w.r.t the popularity of the PoIs. To this end, we make a sample of PoIs which are frequently visited by trajectories in the training set. We consider the top-10 most frequent PoIs in three datasets. The trajectories in the test set passing through one of these frequent PoIs in the tail, make a set of queries including frequently visited PoIs. The rest trajectories in test set make a set containing rarely visited PoIs. To be as fair as possible we evaluate the PROB baseline in the same way.

As we can see in Table 6.4, there is a significant difference among frequent (F) and rare (R) PoI prediction results. For Pisa, PROB predicts the frequent PoIs with a 22.27% in terms of Success@1, while the accuracy of predicting the rarely visited PoIs is so low, 1.12%. While, in the case of Florence and Rome, the performance of predicting rare PoIs is better with respect to the accuracy of frequent PoIs. Although, it is still low with 2.45% and 8.93%

for rare PoIs against 11.95% and 21.69% for frequent PoIs in Florence and Rome, respectively.

As we can see, the proposed method MsTraj offers a significant improvement for both frequent and rare PoIs. We can drive that MsTraj can predict the frequent PoIs better than rare PoIs with high quality, while still can predict the rare PoIs with a good accuracy like 44.94%, 44.77% and 71.55% in Pisa, Florence and Rome, respectively.

The degree and degree centrality of PoIs To evaluate our method with respect to the degree and degree centrality, we make two samples of the test set of trajectories in each city. We consider the top-10 high degree PoIs in three datasets. The trajectories in the test set passing through one of this high degree PoIs in the tail, make a set of queries including high degree visited PoIs. The remaining trajectories in the test set make a set containing low degree PoIs. Similarly, we evaluate our method w.r.t the degree centrality in the same way for making samples of high and low degree centrality PoIs. As a result, Table 6.5 shows that MsTraj always outperforms Prob in both low degree (degree centrality) and high degree (degree centrality). While we expect that Prob predicts the high degree PoIs with high quality, we can see that Prob captures the high degree PoIs in Florence with 3.82% accuracy. On the other hand, the results show that MsTraj predicts the high degree PoIs with 85.10%, 92.89% and 73.61% accuracy in Pisa, Rome, Florence, respectively. Indeed, the quality of both the prediction models is similar regarding degree and degree centrality. This is due to the specific property of the graph, as each path of the graph could be a trajectory in the training set. Thus, the degree and the degree centrality of nodes of graph G are identical.

Dataset	Predictor	Success@1 %			
		Degree		Degree Centrality	
		HD	LD	HDC	LDC
Pisa	Prob	20.21	11.42	20.21	11.42
	MsTraj	85.10	54.28	85.10	54.28
Rome	Prob	9.59	13.22	9.59	13.22
	MsTraj	92.89	74.05	92.89	74.05
Florence	Prob	3.82	4.61	3.82	4.61
	MsTraj	73.61	48.63	73.61	48.63

Table 6.5: Effectiveness of the proposed method on predicting the next PoI of the currently visited PoI with a high degree (HD) or low degree (LD) and with a high degree centrality (HDC) or low degree centrality (LDC)

The closeness of the PoIs To investigate the ability of the method to predict the PoIs which are far from currently visited PoI, we take two samples of trajectories in the test data. In this case, we have two sets of equal number of trajectories. In particular, one group containing trajectories like $Q = \langle (p_1, t_1), (p_2, t_2) \cdots, (p_{N-1}, t_{N-1}), (p_N, t_N) \rangle$ is such that the Euclidean distance between two last visited PoIs P_N and P_{N-1} is small, while the other group includes trajectories where this distance is large. Table 6.4 contains the results with respect to the geographically closeness of PoIs. As it can be seen, the performance of the MsTraj for predicting both far (GF) and close (GC) PoIs outperforms the performance of Prob, significantly.

Dataset	Length of Trajectory >2 #PoIs of query trajectory	Success@1 %
Pisa	1-PoI	73.50
	2-PoIs	73.50
Rome	1-PoI	80.0
	2-PoIs	80.0
Florence	1-PoI	57.10
	2-PoIs	57.10

Table 6.6: Effectiveness of the proposed method by considering either one PoI (1-PoI) (i.e. (N-1)th PoI) or two PoIs (2-PoI)(i.e. (N-2)th and (N-1)th PoIs) of query trajectory. In this case, we consider only trajectories in the test set with a length of more than 2.

The number of query PoIs Given a query trajectory Q, the number of pairs of $Q.head$ considered to compute the similarity score between Q and other trajectories in training set is an important information for prediction. Thus, we make a sample of trajectories in the test set containing the trajectories with the length more than or equal to three. Let $Q = \langle (p_1, t_1), (p_2, t_2) \cdots, (p_{N-1}, t_{N-1}), (p_N, t_N) \rangle$ be a trajectory in the training set such that $N \geq 3$. For each trajectory like Q in the test set, we illustrate two experiments in the following. One experiment by considering $q = \langle (p_{N-1}, t_{N-1}) \rangle$ in line 65 in Algorithm 6 and another experiment by having $q = \langle (p_{N-2}, t_{N-2}), (p_{N-1}, t_{N-1}) \rangle$ under consideration. The results for both experiments over three cities are shown in Table 6.6. As we can see, the results are constant. This is due to the specific properties of the datasets and the length of trajectories in each dataset. Looking at the distribution of the length of the trajectories in each city plotted in the Figure 6.1, we see that the length of the majority of trajectories in the three datasets is small. Thus, we expect that the length of the trajectories in the training

set within the time interval t_{N-1} is similar to the length of the trajectories within time interval $t_{N-1} + t_{N-2}$. Therefore, the performance of the prediction does not change for different numbers of query PoIs in these datasets. The results show that the model behaves as we expected w.r.t the number of query PoIs. Note that, we only consider the trajectories with the length more than or equal to three, since the number of trajectories with a larger length are negligible (Figure 6.1).

6.4.8 Question Q4: Varying the similarity functions

To show the robustness of the proposed method with respect to the different similarity functions, we consider three kinds of similarity functions as follows:

$$Sim_1(Q,T,t) = \sum_{(p_i,t_i)\in Q[t]:t_i\cap t\neq 0} \frac{|t_i| \times \exp^{-dist(p_i,T,t_i)}}{|t|} \quad (3)$$

$$Sim_2(Q,T,t) = \sum_{(p_i,t_i)\in Q[t]:t_i\cap t\neq 0} \frac{|t_i|/|t|}{1 + dist(p_i,T,t_i)}$$

$$Sim_3(Q,T,t) = \sum_{(p_i,t_i)\in Q[t]:t_i\cap t\neq 0} \frac{|t_i|/|t|}{1 + dist_{euc}(p_i,T,t_i)}$$

For a given pair (p, t_i) and the trajectory T, we define $dist(p, T, t_i)$ and $dist_{euc}(p, T, t_i)$ as the temporal network distance (Definition 3.1) and the temporal Euclidean distance, respectively. Let $T[t_i] = \langle (p_{i_1}, t_{i_1}), ..., (p_{i_n}, t_{i_n})\rangle$. We have: $dist_{euc}(p, T, t_i) = d_{euc}((p.x, p.y), M((p_{i_1}.x, p_{i_1}.y), (p_{i_n}.x, p_{i_n}.y))$, where $M(.,.)$ is the midpoint between two points on the plane. Moreover, $p.x$ and $p.y$ denote the x coordination and y coordination of the PoI p.

We conduct the experiments by considering these three different similarity functions. In particular, we examine the behavior of the algorithm over Sim_1, which assigns a larger contribution to closer part of the trajectories, Sim_2, which assigns an equal contribution to close and far parts of trajectories, and Sim_3, which computes the Euclidean distance between trajectories.

The result of the experiments shows that the performance of the proposed method remains constant by varying similarity functions. In particular, the prediction of next-PoIs, in this case, reaches 67.33%, 53.57% and 77.96%, for Pisa, Florence, and Rome, respectively, for each similarity function, which outperforms competitors. This is because of the characteristics of the trajectories in three datasets. As Figure 6.1 shows, the number of trajectories with a length of more than two is small. Thus, the similarity functions cannot have a different behavior while the distance is computing between the

small number of PoIs of trajectories. The robustness of the method regarding Euclidean distance, confirms the effect of the time interval the user spends in PoIs.

In conclusion, the experimental results show the effectiveness of the proposed method with respect to the different parameters and confirm the role of the temporal aspect of trajectories, in terms of the time interval each user spends at each PoI, to the prediction next-PoI.

6.5 Conclusion

In this chapter, we studied the next-PoI prediction problem (next-PoI), which aims at identifying the PoI that a tourist will visit in the future with highest probability. We introduced a new graph-based method that reflects similar behavior of past tourists, to predict the next movements of a new tourist. We conducted a comprehensive evaluation of our proposed method with respect to the state-of-the-art competitors on three public datasets of movements in Pisa, Rome, Florence. The performance of the methods, evaluated in terms of well-known Information Retrieval metrics, shows that our proposal achieves the best performance outperforming well-known competitors based on machine learning. In addition to analyzing the performance in comparison with the baselines, we evaluated the robustness of our approach by answering some different research questions.

First, by studying the efficiency of the proposed method, we observed that our method always outperforms competitors; thus, we can conclude that the proposed method is efficient for predicting the next-PoI of a tourist.

Second, we evaluated the performance of the method, by varying the time unit used to represent the trajectories. We observed that the accuracy of the method is robust w.r.t to a different time scale (day, month, year).

Third, we studied the quality of the model to predict the rarely visited PoIs, which is a challenge for the baseline (probability-based one), showing that our method can predict these PoIs, effectively while outperforming the baseline. Moreover, regarding the prediction of PoIs in graph with low or high degree, and PoIs that are far from the currently visited PoI by the given tourist, we showed in our experiments that the proposed model enables us to predict the far PoIs and even PoIs with high degree in the graph by significantly outperforming the baseline. In addition, we assessed how the length of the given time interval affects the performance of the model. We observed that the performance is constant for larger time intervals, due to the specific properties of datasets and length of trajectories in each dataset.

Finally, our method is extended to different similarity functions, while

outperforming the competitors with the constant performance.

In conclusion, the experimental results showed the effectiveness of the proposed method with respect to the different parameters and confirmed the role of the structural-temporal aspects of trajectories to predict next-PoI. The approach we proposed can be seen as a building block for more complex applications.The approach we proposed can be seen as a building block for more complex applications. We aim at applying our method to other datasets and applications in future such as predicting the next web-page a user will visit in future in a web-network, or the next item a user will buy in a specific period of time in an e-commerce website.

Chapter 7

Conclusion and Future Work

In this thesis, we proposed and investigated the problem of analyzing the similarity between trajectories defined over a network, taking into account the time intervals each object spends at each node of the network. To the best of our knowledge, this is the first work that investigates the top-k most similar trajectory problem on the graph, that is the problem of, given a query trajectory Q, finding the top-k trajectories similar to Q, where trajectories are sequences of nodes of a graph with the corresponding time intervals. This type of query may bring significant benefits to many popular applications such as web usage mining, friend recommendation, tourism applications, fraud detection, and traffic analysis. Indeed, representing the movements of objects by such lower-dimensional trajectories reduced the overall size of the data and the lower-dimensional indexing challenges.

When facing with such a new type of data formed by network constrained trajectories with the corresponding time information, how to efficiently organize this data to process queries against this data is important. In this thesis, we proposed an indexing structure based on the *interval tree* [25] to store trajectories on the graph.

In particular, we studied the k-MsTraj problem, that is, identifying k trajectories with the maximum similarity score to a given query trajectory within a given time interval. On the top of the spatiotemporal similarity functions that are mostly a linear combination of the independently computed spatial and temporal distances [81, 79, 76, 82], in this thesis, we proposed a structural-temporal similarity function, directly in a single function, by computing linear number of distance computations with no limitation on the trajectories. We exploited the topology of the network, assessing that two trajectories are similar if they pass through nearby nodes at roughly the same time.

To address the k-MsTraj query efficiently, we proposed effective algorithms based on two different approaches, aiming at reducing the number of shortest path distance computations, consequently, optimizing the query processing. We proposed an approximate approach with two variations SHQ and SHQT, by dividing the network into Voronoi groups and shrinking trajectories by considering the centers of Voronoi groups. In addition, we presented two exact alternative methods, namely Ordered-based and SkyLine (followed by an approximate variant of the latter i.e. ApSky). By conducting several experiments over some real networks, we show that the proposed methods consistently outperform the baselines and provide strong evidence of the performance and robustness of our solutions. Two different approximate methods aimed at dramatically reduce the querying answering time, at the potential expense of the quality of the solution with respect to the exact one. By the results of the experiments, generally, we conclude that ApSky works well for larger datasets and has the best performance for small values of k, while more accurate results can be obtained by SHQ and SHQT when considering long trajectories for large values of k.

In addition, we studied the next-PoI prediction problem (next-PoI), which aims at identifying the PoI that a tourist will visit in the future with highest probability. We introduced a new graph-based method that reflects similar behavior of past tourists, to predict the next movements of a new tourist.

7.1 Future Work

As mentioned earlier, trajectory retrieval is one of the most important application for trajectory similarity studies. Extracting relevant information by querying large trajectory datasets, helps to prune irrelevant trajectories. Useful tools include range queries, which aim at determining all trajectories passing through either a given node or a set of multiple nodes, within a given time interval. Thus, the performance of a similarity query processing methods crucially depends upon an efficient range querying. When similarity query processes a dataset containing a large number of trajectories that can traverse a node of the graph multiple times, it would be important that a range query could report only the distinct trajectories traversing the given node within a given time interval. This could be more significant when the query aims at looking at nearby trajectories to a set of nodes (e.g. the trajectories traversing a set of nodes belong to a Voronoi group on a graph, within a time interval).

Therefore, the challenge is to identify the distinct trajectories traversing

one node or a set of nodes within a given time interval, which is also a strong motivation for the following applications:

1. Web mining: In a web network, there is interest in determining the number of distinct users that have accessed a website and identifying the most frequently visited websites and so on.

2. Database tuning: Regarding the monitoring of high-performance database servers may someone be interested in analyzing the number of open sessions in a time interval.

The majority of the existing methods for treating range queries on trajectories are based on the R-Tree [1] and do not support networks with no spatial information. Moreover, they are not able to avoid reporting duplicated trajectories in a given area. In this thesis, considering the time intervals spent by trajectories in the nodes of the graph, we used the Interval tree to index trajectories, described in Section 3.2. By searching through the proposed indexing structure, we are able to identify for a trajectory that how much it spent in a given node, as used to design the SkyLine method described in Chapter 5. While, the query processing based on the Voronoi groups and adjusted indexing structure that are proposed in Chapter 4, would be more efficient if, during the searching process through each node of the query with its corresponding time interval, we only report the distinct trajectories. Therefore, in the future, we would attempt to modify the proposed indexing in this thesis aiming at reporting distinct trajectories traversing a given node within a given time interval.

Therefor, the main problem in our future point of view, which is a building block of the similarity queries, is as follows:

TBN-S query Given a set $\mathcal{T}$ containing trajectories defined on a graph $G = (V, E)$ as described in Chapter 3, for a given set of query nodes S_q and a given time interval $t = [a, b]$, we aim at finding the set of distinct trajectories traversing S_q within t, called TBN-S query. The TBN-S outputs a trajectory T in $\mathcal{T}$ if and only if $\exists(v_i, t_i) \in T$ s.t $t_i \cap t \neq \emptyset$.

Treating the TBN-S query, is challenging, when S_q contains more than one query node. One approach would be searching for each query node, independently, and then merging the output results in a single set. In another approach, we can merge the structures assigned to each node in the query time, thereafter, by a single searching process, we are able to report the discovered trajectories. These both approaches could be costly for a large number of query nodes. Therefore, we aim at studying in this direction in the future.

Bibliography

[1] Indexing moving points* pankaj k. agarwal† lars arge* jeff erickson S.

[2] On planning sightseeing tours with tripbuilder. *Information Processing & Management*, 51(2):1 – 15, 2015.

[3] M. R. Abbasifard, H. Naderi, Z. Fallahnejad, and O. I. Alamdari. Approximate aggregate nearest neighbor search on moving objects trajectories. *Journal of Central South University*, 22(11):4246–4253, 2015.

[4] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *International conference on foundations of data organization and algorithms*, pages 69–84. Springer, 1993.

[5] B. Altaf, L. Yu, and X. Zhang. Spatio-temporal attention based recurrent neural network for next location prediction. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 937–942, Dec 2018.

[6] L. Arge. External memory geometric data structures. *Lecture notes of EEF Summer School on Massive Data Sets, Aarhus*, 2002.

[7] A.-L. Barabasi. The origin of bursts and heavy tails in human dynamics. *Nature*, 435(7039):207–211, 2005.

[8] R. Baraglia, C. Frattari, C. I. Muntean, F. M. Nardini, and F. Silvestri. A trajectory-based recommender system for tourism. In *Proc. AMT 2012*, 2012.

[9] R. Baraglia, C. I. Muntean, F. M. Nardini, and F. Silvestri. Learnext: Learning to predict tourists movements. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, CIKM '13, page 751–756, New York, NY, USA, 2013. Association for Computing Machinery.

[10] H. S. Barbosa, F. B. de Lima Neto, A. Evsukoff, and R. Menezes. Returners and explorers dichotomy in web browsing behavior—a human mobility approach. In *Complex Networks VII*, pages 173–184. Springer, 2016.

[11] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer-Verlag TELOS, 2008.

[12] M. Boukhechba, A. Bouzouane, S. Gaboury, C. Gouin-Vallerand, S. Giroux, and B. Bouchard. Prediction of next destinations from irregular patterns. *Journal of Ambient Intelligence and Humanized Computing*, 06 2017.

[13] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *Proceedings of the 31st international conference on Very large data bases*, pages 853–864, 2005.

[14] I. Brilhante, J. A. Macedo, F. M. Nardini, R. Perego, and C. Renso. Where shall we go today? planning touristic tours with tripbuilder. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, CIKM '13, page 757–762, New York, NY, USA, 2013. Association for Computing Machinery.

[15] I. Brilhante, J. A. Macedo, F. M. Nardini, R. Perego, and C. Renso. Tripbuilder: A tool for recommending sightseeing tours. In M. de Rijke, T. Kenter, A. P. de Vries, C. Zhai, F. de Jong, K. Radinsky, and K. Hofmann, editors, *Advances in Information Retrieval*, pages 771–774, Cham, 2014. Springer International Publishing.

[16] C. Calenge, S. Dray, and M. Royer-Carenzi. The concept of animals' trajectories from a data analysis perspective. *Ecological informatics*, 4(1):34–41, 2009.

[17] V. P. Chakka, A. Everspaugh, J. M. Patel, et al. Indexing large trajectory data sets with seti. In *CIDR*, volume 75, page 76. Citeseer, 2003.

[18] F. Chanchary, A. Maheshwari, and M. Smid. Querying relational event graphs using colored range searching data structures. In *Conference on Algorithms and Discrete Applied Mathematics*, pages 83–95. Springer, 2017.

[19] J.-W. Chang, R. Bista, Y.-C. Kim, and Y.-K. Kim. Spatio-temporal similarity measure algorithm for moving objects on spatial networks. In *International Conference on Computational Science and Its Applications*, pages 1165–1178. Springer, 2007.

[20] L. Chen and R. Ng. On the marriage of Lp-norms and edit distance. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 792–803. VLDB Endowment, 2004.

[21] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM, 2005.

[22] M. Chen, X. Yu, and Y. Liu. Mining object similarity for predicting next locations. *Journal of Computer Science and Technology*, 31(4):649–660, 2016.

[23] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie. Searching trajectories by locations: an efficiency study. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 255–266. ACM, 2010.

[24] V. T. De Almeida and R. H. Güting. Indexing the trajectories of moving objects in networks. *GeoInformatica*, 9(1):33–60, 2005.

[25] M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 2000.

[26] E. W. Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[27] M. I. D. Efficiently. The relational interval tree. 2000.

[28] M. Erwig. The graph Voronoi diagram with applications. *Networks: An International Journal*, 36(3):156–163, 2000.

[29] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of ACM SIGMOD*, pages 419–429, Minneapolis, MN, 1994.

[30] E. Frentzos. Indexing objects moving on fixed networks. In *International Symposium on Spatial and Temporal Databases*, pages 289–305. Springer, 2003.

[31] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis. Algorithms for nearest neighbor search on moving object trajectories. *Geoinformatica*, 11(2):159–193, 2007.

[32] E. Frentzos, K. Gratsias, and Y. Theodoridis. Index-based most similar trajectory search. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 816–825. IEEE, 2007.

[33] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[34] Y. Ge, H. Xiong, C. Liu, and Z.-H. Zhou. A taxi driving fraud detection system. In *2011 IEEE 11th International Conference on Data Mining*, pages 181–190. IEEE, 2011.

[35] Ş. Gündüz and M. T. Özsu. A web page prediction model based on click-stream tree representation of user behavior. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–540, 2003.

[36] E. Gurarie, R. D. Andrews, and K. L. Laidre. A novel method for identifying behavioural changes in animal movement data. *Ecology letters*, 12(5):395–408, 2009.

[37] A. Guttman. *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[38] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems (TODS)*, 24(2):265–318, 1999.

[39] X. Huang, Y. Zhao, C. Ma, J. Yang, X. Ye, and C. Zhang. Trajgraph: A graph-based visual analytics approach to studying urban network centralities using taxi trajectory data. *IEEE transactions on visualization and computer graphics*, 22(1):160–169, 2015.

[40] J.-R. Hwang, H.-Y. Kang, and K.-J. Li. Spatio-temporal similarity analysis between trajectories on road networks. In *International Conference on Conceptual Modeling*, pages 280–289. Springer, 2005.

[41] J.-R. Hwang, H.-Y. Kang, and K.-J. Li. Searching for similar trajectories on road networks using spatio-temporal similarity. In *East European Conference on Advances in Databases and Information Systems*, pages 282–295. Springer, 2006.

[42] S. Ichoua, M. Gendreau, and J.-Y. Potvin. *Planned Route Optimization For Real-Time Vehicle Routing*, volume 38, pages 1–18. 05 2007.

[43] T. Joachims. Training linear svms in linear time. In *Proc. SIGKDD*, New York, NY, USA, 2006. ACM.

[44] B. Jun, B. Hong, and B. Yu. Dynamic splitting policies of the adaptive 3dr-tree for indexing continuously moving objects. In *International Conference on Database and Expert Systems Applications*, pages 308–317. Springer, 2003.

[45] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.

[46] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 593–604. ACM, 2007.

[47] J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel. Lars: A location-aware recommender system. In *Proc. ICDE*, pages 450–461. IEEE, 2012.

[48] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye. Mining periodic behaviors for moving objects. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1099–1108, 2010.

[49] K. Liu, Y. Li, F. He, J. Xu, and Z. Ding. Effective map-matching on the most simplified road network. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 609–612, 2012.

[50] S. Liu, L. Chen, and G. Chen. Voronoi-based range query for trajectory data in spatial networks. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 1022–1026, 2011.

[51] Y. Liu and H. S. Seah. Points of interest recommendation from gps trajectories. *International Journal of Geographical Information Science*, 29(6):953–979, 2015.

[52] C. Lucchese, R. Perego, F. Silvestri, H. Vahabi, and R. Venturini. How random walks can help tourism. In *Proc. ECIR*. LNCS, 2012.

[53] C. Lucchese, R. Perego, F. Silvestri, H. Vahabi, and R. Venturini. How random walks can help tourism. In *European Conference on Information Retrieval*, pages 195–206. Springer, 2012.

[54] W. Luo, H. Tan, L. Chen, and L. M. Ni. Finding time period-based most frequent path in big trajectory data. In *Proceedings of the 2013 ACM SIGMOD international conference on management of data*, pages 713–724. ACM, 2013.

[55] T. N. Maeda, K. Tsubouchi, and F. Toriumi. Next place prediction in unfamiliar places considering contextual factors. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '17, New York, NY, USA, 2017. Association for Computing Machinery.

[56] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis. *R-trees: Theory and Applications*. Springer Science & Business Media, 2010.

[57] R. Mariescu-Istodor, A. Tabarcea, R. Saeidi, and P. Fränti. Low complexity spatial similarity measure of gps trajectories. In *WEBIST (1)*, pages 62–69, 2014.

[58] F. Mehboob, M. Abbas, R. Jiang, A. Rauf, S. A. Khan, and S. Rehman. Trajectory based vehicle counting and anomalous event visualization in smart cities. *Cluster Computing*, 21(1):443–452, Mar 2018.

[59] A. Monreale, F. Pinelli, R. Trasarti, and F. Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 637–646. ACM, 2009.

[60] C. I. Muntean, F. M. Nardini, F. Silvestri, and R. Baraglia. On learning prediction models for tourists paths. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 7(1):8, 2015.

[61] A. Noulas, S. Scellato, N. Lathia, and C. Mascolo. Mining user mobility features for next place prediction in location-based services. In *Proceedings of the 2012 IEEE 12th International Conference on Data Mining*, ICDM '12, page 1038–1043, USA, 2012. IEEE Computer Society.

[62] D. Papadias, Y. Tao, P. Kanis, and J. Zhang. Indexing spatio-temporal data warehouses. In *Proceedings 18th International Conference on Data Engineering*, pages 166–175. IEEE, 2002.

[63] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui. Aggregate nearest neighbor queries in spatial databases. *ACM Transactions on Database Systems (TODS)*, 30(2):529–576, 2005.

[64] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *Proceedings 2003 VLDB Conference*, pages 802–813. Elsevier, 2003.

[65] L. Pappalardo, F. Simini, S. Rinzivillo, D. Pedreschi, F. Giannotti, and A.-L. Barabási. Returners and explorers dichotomy in human mobility. *Nature communications*, 6:8166, 2015.

[66] M. Pennacchiotti, F. Silvestri, H. Vahabi, and R. Venturini. Making your interests follow you on twitter. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 165–174. ACM, 2012.

[67] D. Pfoser and C. S. Jensen. Indexing of network constrained moving objects. In *Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, pages 25–32. ACM, 2003.

[68] D. Pfoser, C. S. Jensen, Y. Theodoridis, et al. Novel approaches to the indexing of moving object trajectories. In *VLDB*, pages 395–406, 2000.

[69] I. S. Popa, K. Zeitouni, V. Oria, D. Barth, and S. Vial. Parinet: A tunable access method for in-network trajectories. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 177–188. IEEE, 2010.

[70] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 71–79, 1995.

[71] S. Šaltenis. *Indexing the positions of continuously moving objects*. Springer, 2008.

[72] J. Sankaranarayanan, H. Alborzi, and H. Samet. Efficient query processing on spatial networks. In *Proceedings of the 13th annual ACM international workshop on Geographic information systems*, pages 200–209, 2005.

[73] H. Seifoddini and M. Djassemi. The production data-based similarity coefficient versus jaccard's similarity coefficient. *Computers & industrial engineering*, 21(1-4):263–266, 1991.

[74] W. Sha, Y. Xiao, H. Wang, Y. Li, and X. Wang. Searching for spatio-temporal similar trajectories on road networks using Network Voronoi Diagram. In *Geo-Informatics in Resource Management and Sustainable Ecosystem*, pages 361–371. Springer, 2015.

[75] S. Shang, L. Chen, C. S. Jensen, J.-R. Wen, and P. Kalnis. Searching trajectories by regions of interest. *IEEE Transactions on Knowledge and Data Engineering*, 29(7):1549–1562, 2017.

[76] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis. Trajectory similarity join in spatial networks. *Proceedings of the VLDB Endowment*, 10(11):1178–1189, 2017.

[77] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis. Parallel trajectory similarity joins in spatial networks. *The VLDB Journal*, 27(3):395–420, 2018.

[78] S. Shang, R. Ding, B. Yuan, K. Xie, K. Zheng, and P. Kalnis. User oriented trajectory search for trip recommendation. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 156–167, 2012.

[79] S. Shang, R. Ding, K. Zheng, C. S. Jensen, P. Kalnis, and X. Zhou. Personalized trajectory matching in spatial networks. *The VLDB Journal*, 23(3):449–468, 2014.

[80] L.-A. Tang, Y. Zheng, X. Xie, J. Yuan, X. Yu, and J. Han. Retrieving k-nearest neighboring trajectories by a set of point locations. In *International Symposium on Spatial and Temporal Databases*, pages 223–241. Springer, 2011.

[81] E. Tiakas, A. N. Papadopoulos, A. Nanopoulos, Y. Manolopoulos, D. Stojanovic, and S. Djordjevic-Kajan. Trajectory similarity search in spatial networks. In *null*, pages 185–192. IEEE, 2006.

[82] E. Tiakas and D. Rafailidis. Scalable trajectory similarity search based on locations in spatial networks. In *Model and Data Engineering*, pages 213–224. Springer, 2015.

[83] R. Trasarti, R. Guidotti, A. Monreale, and F. Giannotti. Myway: Location prediction via mobility profiling. *Information Systems*, 64:350–367, 2017.

[84] A. Vázquez, J. G. Oliveira, Z. Dezsö, K.-I. Goh, I. Kondor, and A.-L. Barabási. Modeling bursts and heavy tails in human dynamics. *Physical Review E*, 73(3):036127, 2006.

[85] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 673–684. IEEE, 2002.

[86] S. Wang, Y. Wang, J. Tang, K. Shu, S. Ranganath, and H. Liu. What your images reveal: Exploiting visual contents for point-of-interest recommendation. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 391–400, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[87] J.-I. Won, S.-W. Kim, J.-H. Baek, and J. Lee. Trajectory clustering in road network environment. In *Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on*, pages 299–305. IEEE, 2009.

[88] Y. Xia, G.-Y. Wang, X. Zhang, G.-B. Kim, and H.-Y. Bae. Spatio-temporal similarity measure for network constrained trajectory data. *International Journal of Computational Intelligence Systems*, 4(5):1070–1079, 2011.

[89] H. Xu, P. Wu, J. Wei, Z. Yang, and J. Wang. A meta-path-based recurrent model for next poi prediction with spatial and temporal contexts. In J. Shao, M. L. Yiu, M. Toyoda, D. Zhang, W. Wang, and B. Cui, editors, *Web and Big Data*, pages 219–235, Cham, 2019. Springer International Publishing.

[90] K. Yang and J. Zhu. Next poi recommendation via graph embedding representation from h-deepwalk on hybrid network. *IEEE Access*, 7:171105–171113, 2019.

[91] M. Ye, P. Yin, W.-C. Lee, and D.-L. Lee. Exploiting geographical influence for collaborative point-of-interest recommendation. In *Proc. SIGIR*. ACM, 2011.

[92] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings 14th International Conference on Data Engineering*, pages 201–208. IEEE, 1998.

[93] H. Ying, J. Wu, G. Xu, Y. Liu, T. Liang, X. Zhang, and H. Xiong. Time-aware metric embedding with asymmetric projection for successive poi recommendation. *World Wide Web*, 22(5):2209–2224, 2019.

[94] H. Yuan and G. Li. Distributed in-memory trajectory similarity search and join on road network. In *2019 IEEE 35th international conference on data engineering (ICDE)*, pages 1262–1273. IEEE, 2019.

[95] D. Y. Zhang, D. Wang, H. Zheng, X. Mu, Q. Li, and Y. Zhang. Large-scale point-of-interest category prediction using natural language processing models. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1027–1032, Dec 2017.

[96] S. Zhao, T. Zhao, I. King, and M. R. Lyu. Geo-teaser: Geo-temporal sequential embedding rank for point-of-interest recommendation. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, page 153–162, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[97] K. Zheng, B. Zheng, J. Xu, G. Liu, A. Liu, and Z. Li. Popularity-aware spatial keyword search on activity trajectories. *World Wide Web*, 20(4):749–773, 2017.

[98] V. W. Zheng, Y. Zheng, X. Xie, and Q. Yang. Towards mobile intelligence: Learning from gps history data for collaborative recommendation. *Artificial Intelligence*, 184:17–37, 2012.

[99] Y. Zheng and X. Xie. Learning travel recommendations from user-generated gps traces. *ACM TIST*, 2(1):2, 2011.

[100] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, and G. Sun. A general boosting method and its application to learning ranking functions for web search. *ANIPS*, 20:1697–1704, 2007.

[101] J. Zhu, W. Jiang, A. Liu, G. Liu, and L. Zhao. Effective and efficient trajectory outlier detection based on time-dependent popular route. *World Wide Web*, 20(1):111–134, 2017.