Indexing, Querying, Prediction, and Integration for Network-constrained Moving Object Databases

by

MOHAMMADHANI FOULADGAR

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

2018

To my beloveds

ACKNOWLEDGEMENTS

I would like to thank my supervising professor Dr. Ramez Elmasri for constantly motivating and encouraging me, and also for his invaluable advice during the course of my doctoral studies. I wish to thank my committee members Dr. Gautam Das, Dr. Leonidas Fegaras, and Mr. David Levine for their interest in my research and for taking time to serve in my dissertation committee.

I am grateful to all the professors who taught me during my graduate studies in University of Texas at Arlington. Finally, I would like to express my deep gratitude to my family for their sacrifice, encouragement, patience and inspiration during my PhD work. I am also extremely grateful to several of my friends who have helped me throughout my career.

April 10, 2018

ABSTRACT

Indexing, Querying, Prediction, and Integration for Network-constrained Moving Object Databases

Mohammadhani Fouladgar

The University of Texas at Arlington, 2018

Supervising Professor: Ramez Elmasri

The emergence and presence of satellites and GPS devices have led to the creation of a huge amounts of spatial and spatio-temporal data, which had significant effects on creating new applications to analyze and mine these data. In this regard, a lot of research has been done on moving objects databases as a part of spatial and spatio-temporal databases. In this dissertation, we focus on those moving objects that are not allowed to move in all directions freely, but they (almost) always are restricted to travel on a specific network. One of the most popular example of these moving objects are vehicles that are supposed to travel on a Road Network. This kind of databases are called Network-constrained (or Fixed-network) Moving Object databases. First, we formalize Network-constrained Moving Object databases, and we come up with a Data Model, Data Schema, and Query Schema. Then, we introduce a data structure to index these kinds of databases. We also present a Traffic Congestion Prediction tool by using Deep Artificial Neural Network. Map Integration, Map

Tracking, Map Integrity are other applications in this area that we consider in this dissertation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In this chapter, we first introduce the area of this dissertation research, known as Moving Object Databases (MOD), and our research contributions to MOD in Section 1.1. Then, in Section 1.2, we discuss our research contribution to the area of Traffic Congestion Prediction. Then, in Section 1.3, we give an outline of the remaining chapters of the dissertation in Section 1.3.

1.1 Moving Object Databases (Modeling, Queries, Indexing)

The main purpose of Spatio-Temporal database systems is combining the spatial and temporal features of data. Almost all spatio-temporal applications - such as mobile communication systems, traffic control systems, and GIS with moving objects - have a common basis, which is the requirement to handle both space and time characteristics of the data [9].

With the advent of GPS and wireless devices, as well as portable computing platforms, managing and analysis of the moving objects data are becoming a prominent research area. Moving Objects Database (MOD) are a type of spatio-temporal database that contains information about moving objects and their locations over-time. The key difference that makes this kind of data distinguished from other kinds is the essence of the spatio-temporal nature of this kind of data. In the other words, the locations of moving objects, such as persons (with GPS cell phones), vehicles, ships, and aircrafts, are intensely time-dependent. Therefore, storing and tracking

of the dynamic locations of moving objects needs a special consideration. Hence, recently, significant research focused on the modeling of moving objects.

The moving objects are not limited to vehicles, ships, and aircrafts, rather, hurricanes or oil spills are other instances of moving objects. However, in this work, we consider those moving objects that move in a specific network, such road (traffic) network in terms of vehicles. Thus, throughout this work, the phrase "moving objects" refers to "network-constrained moving objects".

The first step for storing, managing, tracking, and analyzing of moving objects is modeling these data. A comprehensive data model contributes to store these data in Moving Object Databases (MODs) such that processing on them is straightforward. In addition, querying on moving objects data is another substantial issue that comes in the play. For example, in a Police station database, a typical query would be to locate a police car that is currently less than half mile from Adam's Market Complex, Dallas, TX (where assistance is needed). These types of queries can be requested by a user associated with a moving object, or by a stationary user. Applications with these characteristics are referred to as MOD applications, and the queries are known as MOD queries [10].

Furthermore, For queries on large numbers of moving objects, a key issue would be to have efficient indexing structures. These indexing structures, which usually use variations of R-tree [11], assume moving objects can move freely in all directions. However, there are many applications where the moving objects are supposed to move in a specific fixed networks. For instance, vehicles should move in fixed road network. These kinds of moving objects data can be indexed by a Network-Constrained indexing method (also called Fixed-Network indexing method). These indexing structure typically have two layers. The upper layer R-tree indexes the Network (static spatial data). Thus the leaf nodes in upper layer R-tree are line segments (highways, roads,

paths, etc.). For each leaf node in the upper tree, there is an R-tree that stores and indexes time intervals for objects moving along the line segment corresponding to the leaf node [8, 7, 6]. Although Network-Constrained indexing methods are efficient in responding to Spatio-Temporal queries, they usually are not efficient in answering pure Temporal queries, which is an important type of query on Spatio-Temporal data, or when a specific moving object id is also part of the query conditions. For example, find the location of moving object with id of *mid* during a particular time period of $[t_s, t_e]$. In such cases, Network-constrained indexing methods have to scan the entire database to retrieve the result.

In addition, The Network-constrained indexing methods are not able to answer the Strict-path queries efficiently. This query type supports path-based analysis, where trajectories must follow *all* edges in the path. In the following, we discuss our contributions regarding Network-constrained MOD.

1.1.1 Dissertation contributions to Formalization of Network-constrained Moving Object databases

In this contribution, we first categorize the various types of Network-constrained moving object queries.We then propose benchmarks that can be used to compare the performance of systems and indexing schemes that are proposed for handling these types of queries. Network-constrained moving objects are objects that move in a specific network, such as vehicles that are constrained to move in a road (traffic) network.

Our query categories are based on the Network-constrained moving object model presented by [12, 13, 8]. We formally define comprehensive categories of typical queries, based on whether the conditions involve space (point versus region), time (point versus interval), and object id. The categories are based on the various com-

binations of these features. We describe the types of queries as Relational Calculus expressions, based on the query constraints. We focus on three main constraints: Spatial constraints, Temporal constraints, or/and moving object ID constraints. For each types of query, we identify the types of results, and give examples to clarify the query types. This work can define a benchmark for the performance of different types of systems and indexes that are designed to answer queries on Network-constrained moving objects data. Certain indexes/systems may work well for some query categories but perform poorly for other types of queries.

1.1.2 Dissertation contributions to Moving Object Databases querying and indexing

This work describes a new Network-constrained Moving objects indexing structure, which extends the state-of-the-art for this kind of data.The indexing structure we propose is called Temporally Enhanced Network-Constrained R-tree (TENC R-tree), which solves the shortcomings in other Network-Constrained access methods like the FNR-tree [7], MON-tree [6] and UTR-tree. These existing indexing methods are designed to store and retrieve the moving objects based on spatial features, followed by their temporal ones. They are generally not efficient when a query has only temporal constraints, or when a specific moving object id is also part of the query conditions. In such cases, existing methods have to scan the entire database to retrieve the result. Furthermore, the aforementioned methods are not efficient in processing Strict-path query, which is a query type that retrieves trajectories that follow *all* the edges in the queried path [14].

Our proposed TENC R-tree index allows good performance for almost all types of queries on moving objects in a constrained network, whether the constraints are spatial, temporal, or based on object id. Also, the TENC R-tree outperforms other

access methods on the case of *Path queries*. Our experiments show the performance has been improved by 10 to 100 times for such queries.

1.2 Traffic Congestion Prediction

Traffic congestion leads to extra gas emissions and low transportation efficiency, and it wastes a lot of individuals' time and a hunge amount of fuel. Diagnosing congestion and building a pattern for predicting traffic congestion has been regarded as one the most important issues as it can lead to informal decisions on the routes that motorists take, and on expanding road networks and public transport. Research to predict traffic congested spots, especially in urban areas is thus very important.Typcally, congestion prediction can be used in Advanced Traffic Management Systems (ATMSs) and Advanced Traveller Information Systems in order to develope proactive traffic control strategies and real-time route guidance.[15]

In the last decades, concepts of traffic bottleneck and congestion propagation have been considered in many studies. Although most of these originate from Civil Engineering and Urban Transportation studies, the advent of super powerful computers and complex algorithms, traffic management and traffic flow prediction to become an interdisciplinary study.

In this regard, there have been various efforts to predict short-term traffic flow prediction, including mathematical equations [16, 17], simulation techniques [18], or statistical and regression approaches. However, traffic flow is based on individuals' decisions, which more likely can be modeled by Artificial Neural Network the best. In other words, traffic flows are made by individuals' decisions based on their knowledge about currenct traffic and their experiences about past traffic flows, which can be modeled by Artificial Neural Network. Using Neural Network for modeling traffic flow and congestion prediction came to the picture in 1993 in [19]. This work propose

a network consisting of one *input* layer, one *hidden*, and one *output* layer. Although this structure was proven to perform well in many applications for predicting traffic flow and travel time and estimation, it was not efficient in lots of other, because of the simple structure. Therefore, some research uses a Neural Network, initially, to extract traffic flow patters (clustering), and then based on each pattern, they come up with a proper model to predict traffic flow [20, 21, 22]. In this trend, [15] different predictors have different performance for various particular time periods. In other words, each predictor can have a super performance only in a particular time period. Therefore, they combined several predictors together as module to have a better performance for longer time periods.

The data regarding Traffic Flow and Traffic Congestion are two instances of Spatio-temporal data. They embady a location (Spatial Feature) and a time (Temporal feature). Besides, as we already mentioned, traffic flow and traffic congestion are based on human actions [23]. In [24], the authors propose a fully automatic deep model for human-action-based spatio-temporal data. This model first utilizes Convolutional Neural Network model (CNN) to learn the spatio-temporal features. Then, in the second part of this model, they use the output of the first step to train a recurrent neural network model (RNN) in order to classify the entire sequence. [24] does not mention traffic issues as one of the possible applications of their work, however it seems promising to make some model, which is inspired by their model, to predict traffic flow and congestion.

Dissertation contributions to Urban Traffic Congestion Prediction

In this contribution, we try to predict the traffic flow of Traffic Network points, where we do not have any historical data about them, based on the traffic patterns of Traffic Network points. Therefore, our contributions are as follows:

1. We formally define the traffic flows prediction concepts.
2. We introduce a normalized data representation, which can be used in Neural Network algorithms, or other methods.
3. We present a Deep Convolutional Network, which can be able to learn traffic flow of different traffic points.
4. Then, we present a Recurrent Neural Network, which, apart from its structure, can do the same as Convolutional Network.
5. Both of these models are able to predict n-level traffic prediction for different points of the traffic (e.g. Quiet, light traffic, heavy traffic, congested, etc.)
6. They also put up the predicted average speeds on different points of the traffic network based on the speed limits in that point (e.g. 0.65 of speed limit).
7. Then, we present a Recurrent Neural Network

We propose a decentralized deep learning-based method where each node accurately predicts its own congestion state in real-time based on the congestion state of the neighboring stations. Moreover, historical data from the deployment site is not required, which makes the proposed method more suitable for newly installed stations. In order to achieve higher performance, we introduce a regularized euclidean loss function that favors high congestion samples over low congestion samples to avoid the impact of the unbalanced training dataset. A novel dataset for this purpose is designed based on the traffic data obtained from traffic control stations in northern California. Extensive experiments conducted on the designed benchmark reflect a successful congestion prediction.

1.3 Dissertation Organization

In Chapter 2, We introduce a data model for moving objects data. Then, we cover the Location Update Policies. Besides, we define comprehensive categories

for typical queries on moving objects data, based on query constraints. This subsection also describes each of query categories by Relational Calculus expressions. Furthermore, we briefly compare some of the leading moving object indexing methods considering aforementioned query categories. Finally, we conclude and review future work.

In Chapter 3, we propose an indexing structure, called Temporally Enhanced Network-Constrained R-tree (TENC R-tree), which solves the shortcomings in other Network-Constrained access methods like FNR-tree [7], MON-tree [6] and NDTR-tree. We provides the background to our study and reviews previous work for indexing Spatio-Temporal data. Then, we introduce the Structure Model we used for generating our indexing method. In this subsection we go over NDTR-tree [25, 26, 8] as the base structure for our idea. Besides, we introduce our indexing method, TENC R-tree, and we propose algorithms for inserting and searching by using this indexing method. In the following, we experimentally evaluate our proposed index structure and compare it to NDTR-tree. Finally, we conclude and review our future work.

In chapter 4, we propose a decentralized deep learning-based method. we start our work by some preliminaries in section. Then, we define all the main traffic flow concepts and then bring up the problem we are going to solve. We also, introduce two deep network model, and describe their structures broadly. In section the following, we explain our models and methods in more details. Then, we experimentally evaluate our proposed prediction models and compare them with more simple models.

In Chapter 5, we talk about polylines similarity. Polylines similarity is a very fundamental concept when dealing with Spatial, Spatio-temporal, and Moving object databases. One of Polylines similarity's applications is *Map integration and verification*. Map Integration and Verification is the process of matching two distinct topological datasets that present the same road network. Another application of polylines

similarity can be called *Map integrity*, where we have the topological information for a road network as well as the supper accurate GPS data generated by Moving Objects traveling on the road network, and we need to evaluate the accuracy of topological data by calculating the error between the road coordinates and the GPS data from the moving vehicles. Also, another application can be called *Map Matching*, where we correspond the location points generated by GPS devices associated with moving objects to the road network to determine which road they are traveling (or traveled) on. In this chapter, we concentrate on a very intuitive measure called Frechet distance with superior quality in theory and practice to solve *Map Matching*. The Frechet distance is defined as the minimal length of a leash connecting to a dog on one trajectory with its owner on a second trajectory, both never moving backwards.

Finally, Chapter 6 summarizes the contributions made in this dissertation.

CHAPTER 2

Formalization of Network-constrained moving object queries Format

In this chapter, after having an introduction about Network-constrained Moving Object Databases in Section 3.1, we introduce the data model for moving objects data in Section 2.2. In Section 2.3, we describe a (relational) database schema for Network-constrained Moving Object Data based on the definitions we proposed in Section 2.2. In Section 2.4, we cover the Location Update Policies. Section 2.5, defines comprehensive categories for typical queries on moving objects data, based on query constraints. This section also formalizes each of the query categories by Relational Calculus expressions. In Section 2.6, we describe a relational SQL query schema for the Network-constrained moving object databases based on the database schema we introduce in Section 2.3. In Section 2.7, we briefly explain how query types can play a benchmarking role in moving object systems and indexes. In Section 2.8, we summarize and conclude the research contribution presented in this chapter.

2.1 Introduction

With the advent of GPS and wireless devices, as well as portable computing platforms, managing and analysis of the moving objects data are becoming a prominent research area [27, 28]. The key difference that makes this kind of data distinguished from other kinds is the spatio-temporal essence of this kind of data. In other words, the locations of moving objects, such as vehicles, ships, and aircrafts, as well as people with cell phones, are intensely time-dependent. Therefore, storing and tracking the dynamic locations of moving objects needs special consideration. Hence,

recently significant research focused on the modeling of moving objects [12, 13, 29, 8]. In [29], the authors introduce a data model for trajectories, and discuss the moving object databases and queries. This paper [29] proposes two types of data models for trajectories, namely, *trajectories* and *trajectory samples*. The former trajectory data are time-parameterized curves in plane $\mathbf{R}^2$, where $\mathbf{R}$ is the set of real numbers. One the other hand, *trajectory samples* are finite sequences of time-space points. Then, it describes an efficient way of modeling uncertainty via *beads* for trajectory samples. A *bead* is all the possible trajectories between two consecutive time-space points, considering given speed bounds. The most significant distinction between the work proposed in [29] and what we will discuss is that we introduce a data model for the *Network-constrained moving object databases* rather than freely-moving objects. Network-constrained moving objects are the ones that move in a specific network, such as road (traffic) networks in terms of vehicles or railways in terms of trains. Conversly, the work presented in [29] models moving objects that can move freely in all directions without any constraint, such as hurricanes, oil spills, and animals. However, throughout this work, the phrase "moving objects" refers to the "network-constrained moving objects".

The first step for storing, managing, tracking, and analyzing of moving objects is modeling these data objects and the constrained network. A comprehensive data model contributes to store these data in Moving Object Databases (MODs) such that processing on them is more straightforward. Querying, storing, and indexing of moving objects data are other substantial issues that come into play in these databases.

To explore these research issues, we introduce a moving object database model. Then, based on that model, we formally define comprehensive categories of typical queries on moving object data. In order to do this, we describe all types of queries

by Relational Calculus expressions, based on the query constrains, such as Spatial constraints, Temporal constraints, or/and moving object ID constraints. For each type of query, we come up with the possible results, as well as an example to clarify the query type. Consequently, this work can be used as a benchmark for future research on different aspects of querying moving objects data, specially, in performance comparison of indexing and access methods.

2.2 Data Model

In this section we define Network and Network-constrained moving object formally. This model closely follows the work in [12, 13, 8], with a few additional concepts and definitions, to formalize all the concepts.

Definition 0 (Moving Object). A moving object is a moving entity that is represented by a time-dependent position point in space, such as people, cars, trucks, airplanes, and ships. Typically a moving object is associated with a GPS or other positional monitoring device. Each moving object is associated with a unique identifier, *mid*. The set of all moving objects in a dataset is MO ={ *mids*}, where *mid* is the object identifier of a moving object.

Definition 1 (Traffic Network). A traffic network, *N*, comprises a set of routes, *R*, as well as a set of junctions, *J*, and is defined as $N = (R, J)$.

Definition 2 (Route). A route of network *N*, denoted by *r*, is defined as:

$$r = (rid, geo, len, ((jid_h, pos_h))_{h=1}^{m})$$

Where *rid* and *len* are *r*'s identifier and length, respectively. *geo* is a polyline that describes *r*. (jid_h, pos_h) is the *h*th junction (see Definition 3) in route *r*, where jid_h is the junction identifier, and pos_h is its position in *r* (see Definition 2.a). Each route has a direction, so a two-way road *r* is represented as two routes: *r*+ and *r*- (two

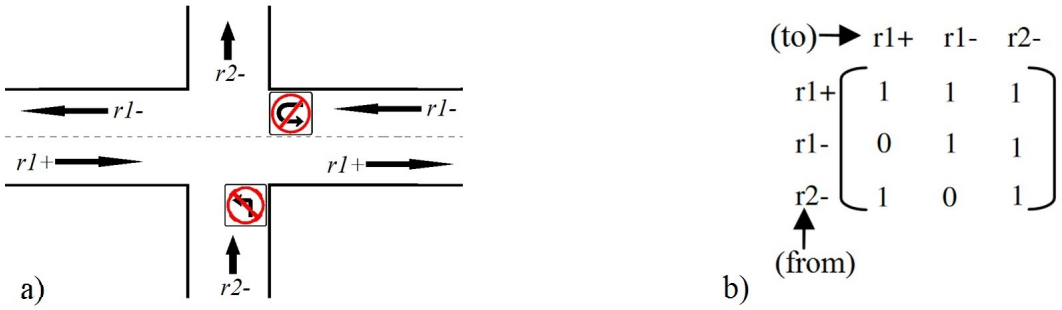

Figure 2.1: a) A junction (road intersection) and the traffic flows inside it. b) Connectivity matrix.

different identifiers). In this case the starting point of $r+$ is typically the ending point of r-, and vice versa.

> **Definition 2.a (Route position).** A position *pos* in a route is a point, which is specified by a fraction value in the normalized range [0, 1]. The beginning point of a route, say r, is assigned to 0, and the end point is assigned to 1. Therefore, a fraction value between 0 and 1 shows positions on r. The distance from the beginning of a route to a point whose position is *pos* will be $pos \times len$.

Definition 3 (Junction). A junction of the traffic network N, denoted by j, is defined as follows:

$$j = (jid, x, y, ((rid_i, pos_i))_{i=1}^{n}, m)$$

Where *jid* is the junction identifier, x and y are the coordinates that show the location of j, (rid_i, pos_i) is the ith route connected by j, which contains the route identifier and position of j on the route (see Definition 2.a), and m is connectivity matrix of j. The connectivity matrix describes traffic flow in the routes connected by a junction. The rows in this matrix denote the flow of the route into the junction, and the columns denote the flow away from the junction to the route (see Figure 2.1). Figure 2.1.a shows an intersection (an instance of junction) with two roads, *r1* and *r2*. Since *r1* is

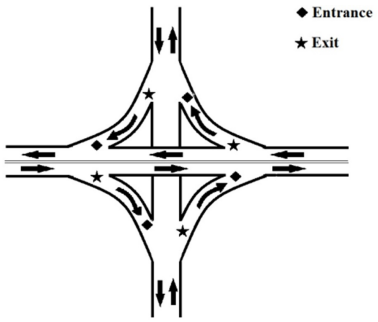

Figure 2.2: Entrance and Exit in highways as an instance of junction

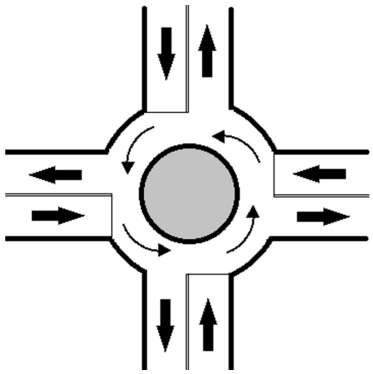

Figure 2.3: Roundabout as an instance of junction

a two-way road, there are two identifier assigned to this road, *r1+* and *r1-*. However, *r2* is one-way, and *r2-* identifies this road. Figure 2.1.b indicates the corresponding connectivity matrix for this intersection. As we observe, a moving object entering to the intersection from *r1+* can continue its path to all directions since there is not any restriction to do U-turn or left-turn. However, an object in *r1-* can either go straight or turn right to *r2-*, but it cannot do a U-turn to go to *r1+*.

It is worth mentioning that junctions can be intersections of streets (see Figure 2.1.a), exit-entrance of highways (see Figure 2.2), roundabouts (see Figure 2.3) and beginning-end point of a route.

Definition 4 (Network Position). A position inside the network N, denoted by *npos*, is defined as follows:

$$npos = \begin{cases} \text{jid}, & \text{if } npos \text{ is a junction of } N \\ (\text{rid,pos}), & \text{if } npos \text{ is a point on a route of } N \end{cases}$$

Definition 5 (Motion Vector). A motion vector mv indicates a description of a moving object's movement at a certain time instant. It is defined as follows:

$$mv = (t,\ v,\ npos)$$

Where v and $npos$ describe the speed and a position (on a route of N) of the moving object in a certain time instant t. In this context, a motion vector does not need a direction because the route direction is known.

Definition 6 (Moving Objects Trajectory). Through location updates, a moving object sends a motion vector to indicate its status in a trip. A sequence of motion vectors sent by the moving object is called a trajectory, *traj*, and is defined as follows:

$$traj = (mv_i)_{i=1}^{n} = ((t_i, v_i, npos_i))_{i=1}^{n}$$

Where v_i and $npos_i$ describe the speed and position of the moving object in ith motion vector of *traj* sent at time instant t_i. As it is a sequence, t_{i+1} ¿ t_i.

Definition 7 (Trajectory units). Two consecutive motion vectors of a trajectory mv_i and mv_{i+1} ($1 \leq i \leq$ n-1) form a trajectory unit, denoted as $\mu(mv_i, mv_{i+1})$. We will use an open time interval $[t_i,\ t_{i+1})$ to refer to the starting time, t_i, and ending time, t_{i+1} , and $npos_i$ and $npos_{i+1}$ to show the starting location and ending location of $\mu(mv_i,\ mv_{i+1})$, respectively. It is worth mentioning that t_{i+1} and $npos_{i+1}$ are not included in $\mu(mv_i, mv_{i+1})$ because we assume open intervals.

Each *mid* usually has an active trajectory unit, plus many non-active (historical) trajectory units. For a non-active trajectory unit, it potentially presents an

infinite number of points $(npos_k, t_k)$ between two motion vectors that are defined as in equation (2.1).

$$t_k \in [t_i, t_{i+1})$$

$$v_{avg} = \frac{npos_{i+1}.pos - npos_i.pos}{t_{i+1} - t_i}$$

$$\left(\begin{array}{c} npos_k.pos = npos_i.pos + (t_k - t_i) \times v_{avg} \\ npos_k.rid = npos_i.rid \end{array} \right) \tag{2.1}$$

As a side note, for calculation of $npos_k$, we assume that the moving object traveled the trajectory unit $\mu(mv_i, mv_{i+1})$ at a steady average speed. This assumption is not unrealistic because when a speed change more than a given threshold occurs, we assume that an update message will be received. Because there cannot be any update message within a trajectory unit, we can say the moving object's speed is (almost) unchanged (i.e. $\frac{npos_{i+1}.pos - npos_i.pos}{t_{i+1} - t_i}$). Therefore, the location of a moving object at time t can be defined as follows:

$$\begin{aligned} mid.NetLoc(t) = & \{npos\langle rid, pos\rangle | (\exists traj \in mid.trajectories) \\ & \wedge (\exists \mu(mv_i, mv_{i+1}) \in traj) \\ & \wedge (\exists (npos, t) \in \mu(mv_i, mv_{i+1})) \\ & \wedge (t_i \leq t) \wedge (t_{i+1} > t) \\ & \wedge (rid = npos_i.rid) \\ & \wedge (pos = npos_i.pos + (t - t_i) \times \frac{npos_{i+1}.pos - npos_i.pos}{t_{i+1} - t_i})\} \end{aligned} \tag{2.2}$$

$$
\begin{aligned}
mid.EucLoc(t) = &\{\langle x, y\rangle | (\exists traj \in mid.trajectories) \\
&\wedge (\exists \mu(mv_i, mv_{i+1}) \in traj) \\
&\wedge (\exists (npos, t) \in \mu(mv_i, mv_{i+1})) \\
&\wedge (t_i \leq t) \wedge (t_{i+1} > t) \wedge \\
&(\exists pos = npos_i.pos + (t - t_i) \times \frac{npos_{i+1}.pos - npos_i.pos}{t_{i+1} - t_i}) \\
&\wedge (x = xPosition(npos_i.rid, pos) \\
&\wedge (y = yPosition(npos_i.rid, pos)\}
\end{aligned}
\tag{2.3}
$$

$mid.NetLoc(t)$ returns the location of moving object mid according to network position, (rid, pos), and $mid.EucLoc(t)$ returns the location according the Euclidian space. $xPosition(rid, pos)$ and $yPosition(rid, pos)$ return x and y coordinates of route rid at position pos. So, for calculation of $npos_k$, we assume that the moving object traveled the trajectory unit $\mu(mv_i, mv_{i+1})$ at a steady speed.

Further, for the active trajectory unit, a moving object must contain the (predicted) current and future position of the moving object. To support this, we have to define the (predicted) future motion vector for the active trajectory unit. Assume mv_n $(t_n, v_n, npos_n(rid_n, pos_n))$ is the last sent update by a moving object. Since we have not received any update message after mv_n , we can say the moving object's speed (v_n) is still (almost) unchanged. Also, we know the moving object has not reached the end of road rid_n, otherwise, we would have received an update message (see section 2.4). Therefore, mv_{active} has the same v_n and rid_n. However we need to calculate future update time (t_{active}) and pos_{active} for mv_{active} as follows:

$$
\begin{aligned}
t_{active} &= t_n + \frac{(1 - pos_n) \times r.len}{|v_n|} \\
pos_{active} &= pos_n + \frac{(t_{active} - t_n) \times |v_n|}{r.len}
\end{aligned}
\tag{2.4}
$$

Definition 8 (Known movement of a moving object). The known movement of a moving object is all trajectories of the moving object defined as follows:

$$mpm = ((mid, \mu(mv_{i,j}, mv_{i+1,j})_{i=1}^{n})_{j=1}^{m}) \tag{2.5}$$

Where *mid* is the moving object identifier, and $\mu(mv_{i,j}, mv_{i+1,j})$ is the ith trajectory unit of the jth trajectory of moving object *mid*.

Definition 9 (Path). A path in a traffic network, denoted by π, is two network positions $npos_s$ and $npos_e$, as well as a sequence of consecutive junctions on the network, $(jid_i)_{i=1}^{n}$, between $npos_s$ and $npos_e$ such that a moving object can travel from $npos_s$ to $npos_e$ on the network without the need to pass any other junctions from the network. A path can be defined as follows:

$$\pi = npos_s + (jid_i)_{i=1}^{n} + npos_e \tag{2.6}$$

In this definition, a path is basically a sequence of consecutive network positions. In general, it is possible to have multiple paths between two consecutive network positions, but we assume the shortest path. Potentially, two consecutive network positions of path π, $npos_i$ and $npos_{i+1}$ ($1 \leq i \leq$ n-1), represent an infinite number of points $npos_k$ that coincide with the shortest path between those two network positions, defined as $\sigma(npos_i, npos_{i+1})$.

2.3 Network-constrained Moving Object Database schema

In this section, we describe a (relational) database schema for Network-constrained Moving Object Data based on the definitions we proposed in section 2.2. There are four main entity types, namely, Moving Objects (*MO*), *Routes*, *Junctions*, and *Trajec-*

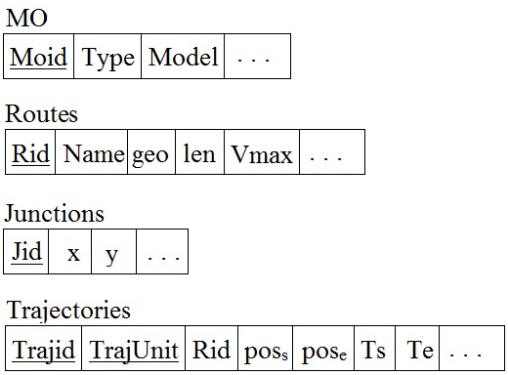

Figure 2.4: Schema of tables for the main entities

tories. Figure 2.4, illustrates the tables for these entities and some of their attributes. As seen, in the *Routes* entity, besides the route identifier (*Rid*), we have other attributes, such as *geo*. *geo* is a spatial attribute with euclidean coordinates of a polyline which represents the route for *Rid*. Recall that from Definition 2, a road can have more than one *Rid*, fox example one *Rid* in each direction. Similarly, in *Junctions* we have the x and the y attributes as the coordination of the junctions. However, the *Trajectories* entity does not include euclidean coordinates. Instead, it comprises the route identifier that the moving object went through, as well as the normalized staring position (pos_s) and the ending position (pos_e). Each record in the *Trajectories* table describes a trajectory unit (see Definition 7). The combination of *Trajid* and *TrajUnit* forms the key for *trajectories*. Put differently, each trajectory has a (unique) *Trajid*. And, for each trajectory, there exists a sequence of n trajectory units, and, consequently, *TrajUnit* values of 1 to n.

Although *MO*, *Routes*, *Junctions*, and *Trajectories* are the main entities in Moving object databases, we need more tables to relate *Routes* to *Junctions*, and *MO* to *Trajectories*. In addition, a table is needed to implement the *Connectivity matrix* (see Definition 3). In Figure 2.5, these complementary tables are illustrated. *Route-Junctions* relates *Routes* and *Junctions* entities. It also indicates the position

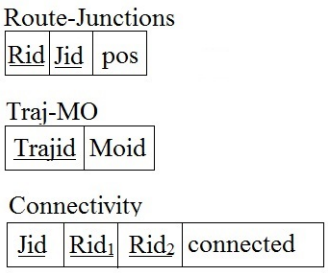

Figure 2.5: Complementary tables

of a junction in a route (*pos*). As mentioned before, *pos* is a real number in [0, 1]. Moreover, *Traj-MO* relates *trajectories* to *MO* entities. As we know, each trajectory is related to just one moving object. Therefore, *Trajid* is the key in this table.

Furthermore, the *Connectivity matrix* is implemented by the *Connectivity* table. Each row in this table represents the connection of routes Rid_1 to Rid_2 in the junction *Jid*. If Rid_1 and Rid_2 are connected (i.e. if a moving object is able to proceed from Rid_1 to Rid_2) the *connected* attribute will be 1, otherwise it will be 0.

2.4 Location update policy

As stated in the previous section, we assume a moving object is associated with a GPS device. A GPS device can generate all the information the moving object needs for putting up a Location Update Message (LUM). In other words, a GPS device can transmit Euclidean coordinates (x, y). By deploying (x, y) and a simple function, (*rid*, *pos*) and v (speed) of the moving object can be calculated. This way we can extract a LUM out of GPS values.

Definition 10 (Location Update Message). A Location Update Message, LUM, is defined as follows:

$$LUM = (mid, t, npos, v) \tag{2.7}$$

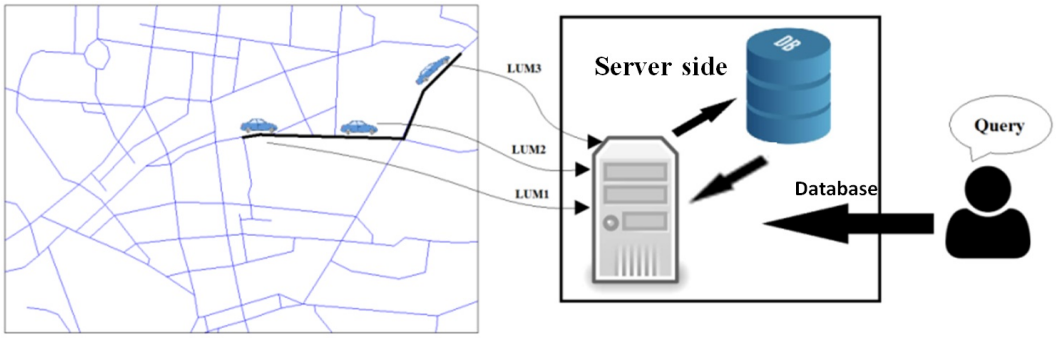

Figure 2.6: Structure of LUM system

Where *mid* is the moving object identifier that sent the message. $npos = (rid, pos)$ is the network position (see Definition 4) of *mid* at time t, and v is the speed.

When a LUM is sent by a moving object, the server will save the information in the database (see Figure 2.6). For instance, suppose that object *mid* sends $LUM = (mid_{LUM}, t_{LUM}, npos_{LUM}, v_{LUM})$ to the server, and, also, assume that the active trajectory unit for mid_{LUM} is $\mu(mv_n, mv_{active})$. At this moment, the server updates the active trajectory unit into $\mu(mv_n, mv_{LUM})$, where $mv_{LUM} = (t_{LUM}, v_{LUM}, npos_{LUM})$. Also, the server inserts a new trajectory unit $\mu(mv_{LUM}, mv_{active'})$ to the database as the new active trajectory unit. $mv_{active'}$ can be generated as we discussed in Definition 7. It should be noted that, sometimes, in a LUM several changes are done on a trajectory. We will discuss this in the following sections.

A related point to consider is that LUMs are supposed to be sent frequently enough to show the trajectory of a moving object. Therefore, there need to be certain conditions that when met the moving object is required to send an LUM to the server. According to [30, 31], there are 3 kinds of location updates: the ID-Triggered Location Update (ITLU), the Speed-Threshold-Triggered Location Update (STTLU), and the Distance-Threshold-Triggered Location Update (DTTLU).

Table 2.1: Comprehensive common Spatio-temporal queries

Query type	Query sub-type	Condition	Result
Coordinate-based query	Window query	Spatial region and Temporal Interval	1) mids + sub-trajectories 2) mids
	Temporal point query	Spatial point and Temporal Interval	1) mids 2) mids + Temporal point
	Time-slice query	Spatial region and Temporal point	1) mids 2) mids + Spatial point
	Point query	Spatial point and Temporal point	mid
	Pure Spacial query	Spatial point	1) mids 2) mids + Temporal point
		Spatial region	1) mids + sub-trajectories 2) mids
	Pure Temporal query	Temporal point	mids + Spatial point
		Temporal interval	mids + sub-trajectories
	k Nearest-neighbor qery	Spatial point and Temporal point	k mids
ID-based Query	-	ID and/or (Spatial region or point) and/or (Temporal interval or point)	1) sub-trajectories 2) Spatial point 3) Temporal point
Path-based Query	Plain	Path	mids + sub-trajectories
	Strict		
Trajectory-based Query (Navigational)	-	ID Spatial region or point Temporal interval or point	1) Maximum speed 2) Average speed 3) Heading
Combined query	What were the trajectories of objects after they left Tucson between 7:00 am to 8:00 am today in the next hours?		

We suppose the moving object continues its trip in the same route with a steady speed until a new LUM is received. Whenever a moving object drives from one route to another, an LUM needs to be sent to relate the changes to the current (active) trajectory. The LUMs that are sent due to a route change are called ID-Triggered Location Update (ITLU). In a similar way, whenever a speed change of a moving object is more than a threshold ψ, a Speed-Threshold-Triggered Location Update (STTLU) must be sent. Distance change is another factor for a location update. According to Definition 7, the current location of a moving object can be approximately calculated, giving the last location update. If the difference between the actual current location and the calculated current location becomes more that a threshold ξ, a Distance-Threshold-Triggered Location Update (DTTLU) will be sent to the server.

2.5 Category of queries

In this section, we study the various types of Network-constrained moving object queries. In order to do this, we first categorize different possible types of Network-constrained moving object queries. Table 2.1 shows a comprehensive categories of typical queries, based on whether the conditions involve space (point versus region), time (point versus interval), and object id. The categories are based on the various combinations of these features. Also, for each type of query, Table 2.1 identifies the type of result.

In the following subsections, we describe the types of queries as Relational Calculus expressions, and give examples to clarify the query types. Subsection 2.5.1 proposes all Query components that can be part of the query condition in Network-constrained moving object queries. Subsections 2.5.2, 2.5.3, 2.5.4, and 2.5.5 formalize the query types by presenting Relational Calculus expressions, and give examples to clarify the query types mentioned in Table 2.1.

2.5.1 Query Components (Q)

In this section we define the comprehensive typical Query components related to Network-constrained moving object data. These components, which are the query conditions, can refer to spatial points (SP), spatial regions (SR), temporal points (TP), and temporal intervals (TI), as well as moving object ID (*mid*). In the case of Path Queries (see 2.5.4) the query component includes a path, π (see Definition 9). We first formally define each of these concepts.

- Spatial Point (SP): This component is a point on the Traffic Network defined by SP = (x, y) that corresponds to a valid *npos* (Definition 4).
- Spatial Region (SR): This component is a *MBR* defined by SR = $(x_{min}, y_{min}, x_{max}, y_{max})$.
- Temporal Point (TP): This component is defined as a time point, TP.

- Temporal Interval (TI): This component is a period of time, TI = (sTP, eTP), eTP > sTP.
- Moving Object identifier *mid*: This component is a Moving Object identifier as defined in Definition 0.
- Path π: This component is a path as defined in Definition 9.

Now we define the query types in Table 2.1 mathematically using the notation of Relational Calculus. The proposed Relational Calculus expressions are based on the Network-constrained moving object model presented in section 2.2.

2.5.2 Coordinate-based Query

In this section, we define all coordinate-based queries by the use of Relational Calculus. Coordinate-based queries [32] are the queries that have spatial features, temporal features, or combination of spatial and temporal features as their query conditions. In the following, we go over various types of this category of queries.

2.5.2.1 Window Query

This type of query is the most common type of moving object query. The query conditions in this type are Spatial Region (SR), within Temporal Interval (TI). *Window* refers to a cuboid in the 3D space (X, Y, T) representing (SR, TI). A possible example of these queries would be "Retrieve all moving object identifiers which were in or passed through Tarrant county, Texas, on June 27, 2016 from 4:00am to 5:00pm". Assume we have query Q(SR, TI). The result for this query is a set of moving object

identifiers *mid* which satisfy the condition (SR, TI). The formal definition for this query is as follows:

$$\begin{aligned} Q(SR,TI) = \{mid | (mid \in MO) \\ \wedge (\exists t_k \in Q.TI) \\ \wedge (mid.EucLoc(t_k) \quad in \quad Q.SR)\} \end{aligned}$$

Another possible result type for *Window query* is a set of moving object identifiers, along with their sub-trajectories with the query condition (SR, TI). For example "Retrieve all moving object identifier which were in or passed through downtown Dallas yesterday, from 8:00am to noon". The formal definition for this query is as follows:

$$\begin{aligned} Q(SR,TI) = \{\langle mid, \{\mu(mv_i, mv_{i+1})_{i=1}^{n}\}\rangle | (mid \in MO) \\ \wedge (\exists traj \in mid.trajectories) \\ \wedge (\mu(mv_i, mv_{i+1}) \in traj) \\ \wedge (\exists (npos_k, t_k) \in \mu(mv_i, mv_{i+1})) \\ \wedge (t_k \in Q.TI) \\ \wedge (mid.EucLoc(tk) \quad in \quad Q.SR)\} \end{aligned}$$

Where $\mu(mv_i, mv_{i+1})_{i=1}^{n}$ is a sequence of sub-trajectory units.

2.5.2.2 Space-slice Query

This type of query retrieves moving object identifiers which have been in a spatial point during a time interval. Therefore the query conditions for this type of query are a Spatial point (SP), along with a Temporal Interval (TI). Space-slice refers to a spatial point SP. A possible example of these queries would be "Retrieve all moving

object identifiers which were or passed *Cooper st.* and *Mitchell st.* intersection, today from 3:30pm to 4:00pm". Assume we have query Q(SP, TI). The result for this query is a set of moving object identifiers *mid* which satisfy the condition (SP, TI). The formal definition for this query is as follows:

$$\begin{aligned} Q(SP,TI) = \{mid | (mid \in MO) \\ \wedge (\exists t_k \in Q.TI) \\ \wedge (traj \in mid.EucLoc(t_k) \quad = \quad Q.SP)\} \end{aligned}$$

Another possible result for *Space-slice query* (SP, TI) is a set of moving object identifiers, along with the time t_k such that the moving object was on SP at t_k ($t_k \in$ TI) . For example "Retrieve all moving object identifiers which left highway I-20 from Eixt 23A, from 3:00am to 3:15am, as well as the exact time". The formal definition for this query is as follows:

$$\begin{aligned} Q(SP,TI) = \{\langle mid, t_k \rangle | (mid \in MO) \\ \wedge (\exists traj \in mid.trajectories) \\ \wedge (\mu(mv_i, mv_{i+1}) \in traj) \\ \wedge (\exists (npos_k, t_k) \in \mu(mv_i, mv_{i+1})) \\ \wedge (t_k \in Q.TI) \\ \wedge (mid.EucLoc(t_k) \quad = \quad Q.SP)\} \end{aligned}$$

2.5.2.3 Time-slice Query

This type of query seeks those moving objects which were in or passed through a specific spatial range at a specific temporal point. Thus, the query conditions for this type of query are a Spatial range (SR), associated with a Temporal point (TP).

A query such as "Retrieve all moving object identifiers which were at JF Kennedy airport area at 1:00pm" is a *Time-slice* query. The formal definition for query Q(SR, TP) is as follows:

$$Q(SR, TP) = \{mid | (mid \in MO) \wedge (mid.EucLoc(TP) \quad in \quad Q.SR)\}$$

However, Time-slice query can return the location of the moving objects in addition to moving object identifier *mid*. For instance, " Retrieve all moving object identifiers which were in or passed through UTA area at 8:00am this morning, and their exact location." The formal definition for this query is as follows:

$Q(SR, TP) =$

$$\{\langle mid, npos_k \rangle | (mid \in MO) \wedge (\exists traj \in mid.trajectories) \wedge (\exists \mu(mv_i, mv_{i+1}) \in traj) \wedge (\exists (npos_k, t_k) \in \mu(mv_i, mv_{i+1})) \wedge (t_k \quad = \quad Q.TP) \wedge (mid.EucLoc(tk) \quad in \quad Q.SR)\}$$

2.5.2.4 Point Query

Point query is a very specific type of moving objects query which asks about moving object(s) in a specific spatial point and at specific temporal point. The query condition for this type of query is in the form of (SP, TP).Assume we have query

Q(SP, TP); the formal definition for this query is as follows:

$$Q(SP,TP) = \{mid|(mid \in MO) \wedge(mid.EucLoc(TP) \quad = \quad Q.SP)\}$$

2.5.2.5 Pure Spatial Query

In all query types that we discussed so far, the Spatial features are associated with Temporal features. However, in Coordinate-based queries, there can exist queries which are pure Spatial (e.g. "retrieve all cabs which had at least one journey in New York city"). It may be observed that *Pure Spatial* Query has two sub-categories: *Pure Spatial* query with Spatial Point (SP) as its query condition, and *Pure Spatial* query with Spatial Range (SR) as its query condition. In the following, we define both of these sub-categories formally.

1. **Pure Spatial Point Query:** Assume we have a *Pure Spatial* query Q(SP). The result of this query can be a set of moving object identifiers. These moving objects have been in Spatial point SP at least once. The formal definition for this query is as follows:

$$\begin{aligned} Q(SP) = \{mid|(mid \in MO) \\ \wedge(\exists traj \in mid.trajectories) \\ \wedge(\exists \mu(mv_i, mv_{i+1}) \in traj) \\ \wedge(\exists (npos_k, t_k) \in \mu(mv_i, mv_{i+1})) \\ \wedge(mid.EucLoc(t_k) = Q.SP)\} \end{aligned}$$

However, given a *Pure Spatial* query Q(SP), sometimes, we look for not only the moving objects, but also the time they were in that location. The formal definition for this query is as follows:

$$\begin{aligned} Q(SP) = \{\langle mid, t_k\rangle | (mid \in MO) \\ \wedge(\exists traj \in mid.trajectories) \\ \wedge(\exists \mu(mv_i, mv_{i+1}) \in traj) \\ \wedge(\exists (npos_k, t_k) \in \mu(mv_i, mv_{i+1})) \\ \wedge(mid.EucLoc(t_k) = Q.SP)\} \end{aligned}$$

2. **Pure Spatial Region Query:** *Pure Spatial* query can ask about a Spatial region (e.g. "retrieve all cabs which had at least once one journey in New York city"). The formal definition for this query is as follows:

$$\begin{aligned} Q(SR) = \{mid | (mid \in MO) \\ \wedge(\exists traj \in mid.trajectories) \\ \wedge(\exists \mu(mv_i, mv_{i+1}) \in traj) \\ \wedge(\exists (npos_k, t_k) \in \mu(mv_i, mv_{i+1})) \\ \wedge(mid.EucLoc(t_k) \quad in \quad Q.SR)\} \end{aligned}$$

A variation of *Pure Spatial Region* queries, returns the moving object identifiers as well as their sub-trajectories that the moving object passed in SR. The formal definition for this query is as follows:

$$
\begin{aligned}
Q(SR) = \{\langle mid, \mu(mv_i, mv_{i+1})_{i=1}^{n}\rangle | (mid \in MO) \\
&\wedge (\exists traj \in mid.trajectories) \\
&\wedge (\mu(mv_i, mv_{i+1}) \in traj) \\
&\wedge (\exists (npos_k, t_k) \in \mu(mv_i, mvi+1)) \\
&\wedge (mid.EucLoc(t_k) \quad in \quad Q.SR)\}
\end{aligned}
$$

Where $\hat{\text{I}}\text{ij}(mv_i, mv_{i+1})_{i=1}^{n}$ is a sequence of sub-trajectory units.

2.5.2.6 Pure Temporal Query

A very common types of moving object queries are *Pure Temporal* queries. Unlike *Pure Spatial* queries, *Pure Temporal* query conditions are based on time conditions only. These types of queries are sub-categorized into Pure Temporal Point and Pure Temporal Interval.

1. **Pure Temporal Point Query:** This type of query retrieves all moving objects' locations at a specific time. Therefore, the query condition is a Temporal point, and the result will be all moving object identifiers and corresponding locations. For example, "Retrieve all police cars' locations today at 3:00pm".

The formal definition for this query is as follows:

$$Q(TP) = \{\langle mid, npos_k\rangle | (mid \in MO) \wedge (\exists traj \in mid.trajectories) \wedge (\exists \mu(mv_i, mv_{i+1}) \in traj) \wedge (\exists (npos_k, t_k) \in \mu(mv_i, mv_{i+1})) \wedge (t_k = Q.TP)\}$$

2. **Pure Temporal Interval Query:** This type of query retrieves all moving objects' sub-trajectories during a specific Temporal Interval. Therefore, the query condition is a Temporal Interval, and the result will be all moving object identifiers and corresponding sub-trajectories. For example, "Retrieve all police cars' sub-trajectories yesterday at from 8:00pm to midnight". The formal definition for this query is as follows:

$$Q(TI) = \{\langle mid, \mu(mv_i, mv_{i+1})_{i=1}^{n}\rangle | (mid \in MO) \wedge (\exists traj \in mid.trajectories) \wedge (\mu(mv_i, mv_{i+1}) \in traj) \wedge (\exists (npos_k, t_k) \in \mu(mv_i, mv_{i+1})) \wedge (t_k \in Q.TI)\}$$

Where $\mu(mv_i, mv_{i+1})_{i=1}^{n}$ is a sequence of sub-trajectory units.

2.5.2.7 *K* Nearest-Neighbor Query

The query condition in this type are Spatial Point (SP), along with Temporal Interval (TI), as well as an integer value k. The result will be k nearest moving

objects to SP in Temporal Interval TI. A possible example of these queries would be "Retrieve 5 nearest ambulances to city hall from 3:00pm to 3:15pm". Assume we have query Q(SR, TI). The result for this query is a set of moving object identifiers *mid* which satisfy the condition (SR, TI). The formal definition for this query is as follows:

$$
\begin{aligned}
Q(k, SP, TP) = & \\
& \{mid_j | (mid_j \in MO) \\
& \quad \wedge D = \{d_i | (mid_i \in MO) \\
& \quad \wedge (d_i = dist(mid_i.EucLoc(TP), SP))\} \\
& \quad \wedge (d_j = dist(mid_j.EucLoc(TP), SP)) \\
& \quad \wedge (d_j \in MIN(k, D))\}
\end{aligned}
$$

Where MIN (k, D) returns the smallest k values in a set D of numbers, and dist(SP_1, SP_2) returns the distance between two spatial points SP_1 and SP_2.

2.5.3 ID-based Queries

All queries that we covered so far are Coordinate-based queries, which have only Spatial and/or Temporal features. Therefore, they retrieve ***all*** moving objects that satisfy the query condition. However, we sometimes want to limit the result to some specific moving objects by adding moving object identifier, *mid*, as part of the query condition. The query types, which have moving object identifier as their query condition, are called *ID query*.

These types of queries are similar to Coordinate-based queries, however they have *ID* as an additional query condition. In the following we define various types of *ID query* formally. ID queries can include all the coordinate-based query types aug-

mented with a particular moving object identifier, *mid*. We only give some examples here.

Assume we have query Q(mid, SR, TI) which means return all sub-trajectories (as the result) of moving object *mid* in Spatial Region *SR* in during Temporal Interval *TI*. As an example, "return all sub-trajectory of *Ambulance 286* during last week in the DFW metroplex". This type of query is defined as follows:

$$\begin{aligned} Q(mid, SR, TI) = \{\langle \mu(mv_i, mv_{i+1})_{i=1}^{n} \rangle | (\exists traj \in mid.trajectories) \\ \wedge (\mu(mv_i, mv_{i+1}) \in traj) \\ \wedge (\exists (npos_k, t_k) \in \mu(mv_i, mv_{i+1})) \\ \wedge (t_k \in Q.TI) \\ \wedge (mid.EucLoc(t_k) \quad in \quad Q.SR)\} \end{aligned}$$

Where $\mu(mv_i, mv_{i+1})_{i=1}^{n}$ is a sequence of sub-trajectory units.

The query Q(mid, SP, TI) means return all Temporal Points (as the result) when moving object *mid* was in or passed through a Spatial Point *SP* during Temporal Interval *TI*. For example "Retrieve the times that car licence number 'BD51' passed *Collins* and *Pioneer Pkwy* intersection during last week". The formal definition for this query is as follows:

$$\begin{aligned} Q(mid, SP, TI) = \{t_k | (t_k \in Q.TI) \\ \wedge (mid.EucLoc(t_k) = Q.SP)\} \end{aligned}$$

Assume we have query Q(mid, SR) which returns all Temporal Intervals (as the result) when moving object *mid* was in or passed through a Spatial Region *SR*. For example "Retrieve the times that car licence number 'BD51' was in or passed through

Houston". The formal definition for this query is as follows:

$$Q(mid, SR) = \{TI | (\forall t_k \in TI) \\ \wedge (mid.EucLoc(t_k) = Q.SP)\}$$

Also, the result for Q(mid, SR) can mean return all sub-trajectories of *mid* (as the result) when moving object *mid* was in or passed through a Spatial Region *SR*. The formal definition for this query is as follows:

$Q(mid, SR) =$

$$\{\langle \mu(mv_i, mv_{i+1})_{i=1}^{n} \rangle | (\exists traj \in mid.trajectories) \\ \wedge (\mu(mv_i, mv_{i+1}) \in traj) \\ \wedge (\exists (npos_k, t_k) \in \mu(mv_i, mv_{i+1})) \\ \wedge (mid.EucLoc(t_k) \quad in \quad Q.SR)\}$$

Where $\mu(mv_i, mv_{i+1})_{i=1}^{n}$ is a sequence of sub-trajectory units.

The query Q(mid, SP) returns all Temporal Points (as the result) when moving object *mid* was in or passed a Spatial Point *SP*. For example "Retrieve the times that car licence number 'BD51' was in or passed *Collins* and *Pioneer Pkwy* intersection". The formal definition for this query is as follows:

$$Q(mid, SP) = \{t_k | (mid.EucLoc(t_k) = Q.SP)\}$$

The result for Q(mid, TP) returns the location that *mid* was (as the result) at Temporal Point *TI*. The formal definition for this query is as follows:

$$\begin{aligned} Q(mid, TP) = \{\langle npos_k \rangle | (mid \in MO) \\ \wedge (\exists traj \in mid.trajectories) \\ \wedge (\exists \mu(mv_i, mv_{i+1}) \in traj) \\ \wedge (\exists (npos_k, t_k) \in \mu(mv_i, mv_{i+1})) \\ \wedge (t_k = Q.TP)\} \end{aligned}$$

As a final example, assume we have query Q(mid, TI) which returns all trajectories of *mid* (as the result) in Temporal Interval *TI*. For example "Retrieve all sub-trajectories of car license number 'BD51' on Monday, May 30, 2016 ". The formal definition for this query is as follows:

$$\begin{aligned} Q(mid, TI) = \{\mu(mv_i, mv_{i+1})_{i=1}^{n} \rangle | (\exists traj \in mid.trajectories) \\ \wedge (\mu(mv_i, mv_{i+1}) \in traj) \\ \wedge (\exists (npos_k, t_k) \in \mu(mv_i, mv_{i+1})) \\ \wedge (t_k \quad \in \quad Q.TI)\} \end{aligned}$$

Where $\mu(mv_i, mv_{i+1})_{i=1}^{n}$ is a sequence of sub-trajectory units.

2.5.4 Path-based Queries

Another type of query has a path, along with a Temporal Interval as its query condition. This type of query is called *Path-based Query*. We discuss two types of path queries in the following:

2.5.4.1 Plain (Partial) Path Queries

In this type of query, the input is a *path*, π, (see Definition 9) and a Temporal Interval [33]. This query returns the set of sub-trajectories which visit at least one edge in the query path, π within the specific time period.The formal definition for this query is as follows:

$$\begin{aligned} Q(\pi, TI) = \{\langle mid, \mu(mv_i, mv_{i+1})_{i=1}^{n}\rangle | (\exists traj \in mid.trajectories) \\ \wedge(\mu(mv_i, mv_{i+1} \exists traj) \\ \wedge(\exists(npos_k, t_k) \in \mu(mv_i, mv_{i+1}) \\ \wedge(t_k \in Q.TI) \\ \wedge(npos_k in \pi)\} \end{aligned}$$

Where $\mu(mv_i, mv_{i+1})_{i=1}^{n}$ is a sequence of sub-trajectory units.

2.5.4.2 Strict Path Queries

In this type of query, the input is a *path*, π, (see Definition 9) and a Temporal Interval [14]. This query returns all sub-trajectories that strictly follow the path,π, within a Temporal Interval defined by t_s and t_e. The result of this query is a set of sub-trajectories that has π as its sub-path, enters π at a time point equal to or later

than t_s, and leaves π at a time point equal to or before t_e [14]. The formal definition for this query is as follows:

$$\begin{aligned} Q(\pi, TI) = \{\langle mid, \mu(mv_i, mv_{i+1})_{i=p}^{q}\rangle | (\exists traj \in mid.trajectories) \\ \wedge(\mu(mv_l, mv_{l+1})_{l=1}^{n} = traj) \\ \wedge(\mu(mv_i, mv_{i+1})_{i=p}^{q} \quad in \quad \mu(mv_l, mv_{l+1})_{l=1}^{n}) \\ \wedge(\exists(npos_p, t_p) = mv_p) \\ \wedge(\exists(npos_q, t_q) = mv_q) \\ \wedge(\pi = mid.path(t_p, t_q)) \\ \wedge(\left[t_p, t_q\right] \quad in \quad Q.TI)\} \end{aligned}$$

Where $mid.path(t_s, t_e)$ returns the path moving object mid travel from t_s to t_e, and it can be defined as follows:

$$\begin{aligned} mid.path(t_s, t_e) = \{(npos_k)_{k=1}^{n} | (mid \in MO) \\ \wedge(\exists traj \in mid.trajectories) \\ \wedge(\exists \mu(mv_i, mv_{i+1})_{i=1}^{m} \in traj) \\ \wedge(\exists(npos_k, t_k)_{k=1}^{n} \in \mu(mv_i, mv_{i+1})_{i=1}^{m}) \\ \wedge([t_1, t_n]\, in\, [t_s, t_e])\} \end{aligned}$$

Where $\mu(mv_i, mv_{i+1})_{i=1}^{n}$ is a sequence of sub-trajectory units, and $(npos_k)_{k=1}^{n}$ is a sequence of network positions.

2.5.5 Trajectory-based Query (Navigational)

This type of query derives information from a trajectory [28]. Typically, dynamic information is not explicitly stored. Therefore, this type of query returns the

dynamic information about a trajectory, e.g. the average or top speed of a moving object in a Spatial Region and Temporal Interval. Thus, the query condition for this type of query includes *mid*, *SR*, and *TI*. And the result will be dynamic information about that moving object in the given spatio-temporal range. The formal definition for this query is as follows:

$$\begin{aligned}\{\langle f([\mu(mv_i, mv_{i+1})_{i=1}^{n}])\rangle | (&\exists traj \in mid.trajectories) \\ &\wedge(\mu(mv_i, mv_{i+1})_{i=1}^{n} \in traj) \\ &\wedge(\exists(npos_k, t_k) \in \mu(mv_i, mv_{i+1})) \\ &\wedge(t_k \in Q.TI) \\ &\wedge(mid.EucLoc(t_k) \quad in \quad Q.SR)\}\end{aligned}$$

2.6 Network-constrained moving object query schema

The formal definition of query types in relational calculus can be formulated as SQL queries. Because of space limitations, we give only one example here. We describe a relational SQL query schema for the Network-constrained moving object databases on the tables we introduced in section 2.3. Having those tables, assume we need to have all the street names that moving object 'χ' traveled on 09-20-2016 from 8:00 am to 9:00 am. Consider the following query:

select distinct Routes.Name

from Routes, Trajectories, Traj-MO

where Routes.Rid = Trajectories.Rid **and**

Trajectories.Trajid = Traj-MO.Trajid **and**

Trajectories.Ts ¿ '09-20-2016 8:00' **and**

Trajectories.Te ¡ '09-20-2016 9:00' **and**

Traj-MO.Moid = 'χ'

This example belongs to the ID-based query category combined with temporal aspects, and is one of the multitude of queries that can be generated on this type of data. Because of lack of space, we do not show other instances for other query types.

2.7 Using query types for benchmarking moving object systems and indexes

An important contribution of our work is the comprehensive categorization of the types of queries that can be commonly used in Network-constrained moving objects systems, and the precise definition of each type of query in a formal manner. The data model proposed in [12, 13, 8] was fully formalized in our work and expanded with a few additional concepts, such as paths in the Network-constrained model.

A very important application of our work is to compare the performance of Network-constrained moving objects systems, indexing methods [34], and approaches to implementation. A previous benchmark has been proposed in [35, 36], but it does not include the comprehensive list of query types that we categorized in this work. This benchmark, which is called BerlinMod, is a benchmark for spatio-temporal database management systems (STDBMS) by the University of Hagen. By considering the query types presented in this work, the queries presented in BerlinMod do not include k Nearest-neighbor queries, Path queries, and Navigational Trajectory-based queries. In particular, categorizing the query conditions for query types based on the three orthogonal features of spatial, temporal, and moving object id is an important contribution of this work.

The reason that benchmarking is very important is that certain proposed indexing methods may work well for certain types of queries, for example most recently developed Network-Constrained access methods (FNR-tree [7], MON-tree [6], UTR-tree [25]) can store and retrieve the state of moving objects according to spatial aspects

of data then on temporal aspects restricted to a specific road segment. Therefore, for pure temporal queries, queries based on moving object identifier, (pure) ID query, path queries, or k Nearest-neighbor queries, the system needs to search all records to retrieve the result.

Our comprehensive formalization and categorization of the common types of queries in Network-constrained moving object systems makes it possible to compare the proposed systems and indexing methods in a more complete manner. Each proposed indexing method can clearly identify the types of queries it performs well, and the types of queries it does not work well. This makes it possible for future users to choose the type of index or system that is needed for the types of queries that the user intends to submit.

2.8 Conclusion

In this chapter, we categorized the various types of Network-constrained moving objects queries, which are objects that move in a specific network, based on the model presented by [12, 13, 8]. Then, we formally defined comprehensive categories of typical queries, based on whether the conditions involve space (point versus region), time (point versus interval), and object id as well as the various combinations of these three features. In order to do this, we formalized the query types by presenting Relational Calculus expressions, based on the query constraints: Spatial constraints, Temporal constraints, or/and moving object ID constraints. For each types of query, we identified the types of results, and gave examples to clarify the query types. Certain indexes/systems may work well for some query categories but perform poorly for other types of queries. Therefore, these formal query definitions can be considered as benchmarks that can be used to compare the performance of systems and indexing schemes that are proposed for handling these types of queries.

CHAPTER 3

Temporally Enhanced Network-Constrained (TENC) R-tree

In this chapter, after having an introduction in Section 3.1 and considering previous work in 3.2, we introduce a Structure Model in Section 3.3, we used for generating our indexing method. In this section we consider the UTR-tree [25, 26, 8] as the base structure for our idea. In section 3.4, we introduce our indexing method, the TENC R-tree, and we propose algorithms for inserting and searching for this indexing method. In section 3.5, we experimentally evaluate our proposed index structure and compare it to the UTR-tree.

3.1 Introduction

The main purpose of Spatio-Temporal database systems is combining the spatial and temporal features of data. Almost all spatio-temporal applications - such as mobile communication systems, traffic control systems, and GIS with moving objects - have a common basis, which is the requirement to handle both space and time characteristics of the data [9].

Similar to other data types, spatio-temporal data are required to be queried efficiently. In this regard, Spatio-temporal access methods (STAMs) provide the necessary tools to query spatio-temporal data [37]. These access methods basically are enhanced variations of the well-known R-tree [11, 38].

Generally there are two broad access method categories for spatio-temporal data:

- Access methods and query processing techniques that involve past (historical) object locations, shape changes, or status values [1, 2, 3]. The data is static and no update to the data is expected.
- Access methods and query processing techniques that involve, in addition to object history, the present and future locations, shapes, or status of objects. The future status of an object can be estimated according to its present status and a time function that predicts how the object changes over time. New object positions are continually updated in this case [25, 26, 8].

Consider a spatio-temporal database that contains information about moving objects and their locations. For example, in a Police station database, a typical query would be to locate a police car that is currently less than half a mile from Adam's Market Complex, Dallas, TX (where assistance is needed). These types of queries can be requested by a user associated with a moving object, or by a stationary user. Applications with these characteristics are referred to as Moving Objects Database (MOD) applications, and the queries are known as MOD queries [10].

For queries on large numbers of moving objects, a key issue would be to have efficient indexing structures. These indexing structures, which usually use R-tree [11], assume moving objects can move freely in all directions. However, there are many applications where the moving objects are supposed to move in a specific fixed network. For instance, vehicles should move in a fixed road network. These kinds of moving objects data can be indexed by a Network-Constrained indexing method (also called Fixed-Network indexing method). These indexing structures typically have two layers. The upper layer R-tree indexes the Network (static spatial data). Thus, the leaf nodes in upper layer R-tree are line segments (highways, roads, paths, etc.). For each leaf node in the upper tree, there is an R-tree that stores and indexes time intervals for objects moving along the line segment corresponding to the leaf node

[8, 7, 6]. Although Network-Constrained indexing methods are efficient in responding to Spatio-Temporal queries, they usually are not efficient in answering pure Temporal queries, which is an important type of query on Spatio-Temporal data, or when a specific moving object id is also part of the query conditions; for example, find the location of moving object *mid* during a particular time period of $[t_s, t_e]$. In such cases, Network-constrained indexing methods have to scan the entire database to retrieve the results.

Besides, the Network-constrained indexing methods are not able to answer the Strict-path queries efficiently. This query type supports path-based analysis, where trajectories must follow *all* the edges in the path. Our indexing structure, the **TENC Rtree**, makes retrieval of trajectories that follow a specific path, i.e. Strict-path queries, efficient.

3.2 Previous work

In this section we consider several prominent STAMs chronologically. All these access methods are based on the well-known R-tree [37, 38]. As a side note, there are a large number of spatio-temporal indexing structures to support spatio-temporal data. However, in this section, we cover the ones that are the ancestors of our indexing structure (based on the presented hierarchy in [38]).

3.2.1 The RT-tree [1]

The RT-tree, which was introduced in 1990, is the first effort for using R-tree for indexing spatio-temporal data. It is actually a regular R-tree to which temporal information is added to the nodes. Therefore, the processing of temporal queries requires considerable computational effort.

The RT-trees couple time intervals with spatial ranges in each node of the tree structure. In other words, an object that is inserted at time t_x has its temporal extent initialized to $[t_x, t_x)$. In every time instance, this temporal extent is increased as long as the spatial status of the object remains unchanged. When a spatial change happens at t_y, the current extent is closed as $[t_x, t_y)$ and a new entry is created and its temporal extent is initialized to $[t_y, t_y)$.

3.2.2 The 3D R-tree [2]

The 3D R-tree was introduced in 1996. This access method considers time as an additional dimension. Therefore in 3D R-tree, data are represented as three-dimensional boxes instead of regular two-dimensional rectangles The index entry contains $X \times Y \times T$ $([x_{min}, x_{max}], [y_{min}, y_{max}], [t_{start}, t_{end}])$. The shortcoming for this index structure is that both ends of $[t_{start}, t_{end}]$ need to be known. Thus, for current status (now) of moving objects whose end time is not known this approach does not work well.

A possible solution for this problem can be to show now by a time instant so far in the future. Although this approach seems to be effective, it leads to extensive boxes and causes poor performance.

3.2.3 The 2+3 R-Tree [3]

As mentioned before, the shortcoming for 3D R-tree is the open ended time intervals, which show the current spatial status (*now*) of objects. One approach, which is proposed to overcome this problem is to maintain a pair of two R-trees:

- A 2D R-tree for representing the current spatial information about objects.
- A 3D R-tree for representing the past information about objects.

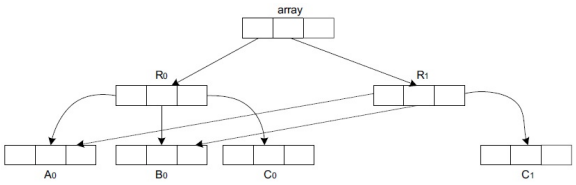

Figure 3.1: HR-tree

3.2.4 The Historical R-tree (HR-tree) [4, 5]

This access method is based on creating a new R-tree for each update. However creating and keeping the whole R-tree for each update is neither practical nor efficient. To overcome this problem, instead of creating a new R-tree physically, for each update a new R-tree is created logically. In other words, HR-tree is based on the overlapping technique. To implement HR-tree, an array is used which is pointing the roots of R-trees. For each update, updated nodes are created. The new tree contains new nodes, but points to unchanged old nodes. This Rtree is now actually a collection of independent trees, but it is an acyclic graph. Figure 3.1 clarifies the structure of HR-tree.

Although HR-Tree is efficient for time slice queries, it is not as efficient as 3D R-tree and 2+3 R-tree for time interval queries [4]. Furthermore, when the number of objects that have location change is not small for each time instance, a large number of nodes need to be regenerated. Therefore the common nodes between the new tree and the previous one would be low, and consequently it leads to a large structure.

3.2.5 Network-Constrained R-tree Category [6, 7, 8]

Network-Constrained R-trees assume that, in most applications, objects cannot move freely in all directions, but they are supposed to move in a specific constrained network. For example, vehicles as instances of moving objects have a restriction to move in Road Networks. Aircrafts, despite general belief, must fly in specific air

corridors. The idea of Network-Constrained moving object indexing was proposed in 2003 by the **FNR-tree** (Fix-Network R-tree) [7] as an access method for fleet management applications, and improved upon in the **MON-tree** in 2005 [6]. In 2008, several prominent Network-Constrained moving object indexing, namely, **GStree** [39], **PPFI** [40], and **UTR-tree** [25] were proposed. GStree is based on constant update interval. PPFI and UTR-tree can deal with not only the historical location of moving objects, but also their current and near future location information.

All Network-Constrained R-tree variations are two-stage access methods. The upper layer R-tree indexes the Network (static spatial data). Thus the leaf nodes in upper layer R-tree are line segments (highways, roads, paths, etc.). For each leaf node in the upper tree, there is an R-tree that stores and indexes time intervals for objects moving along the line segment corresponding to the leaf node [6, 7, 8].

At the beginning, there are no lower layer R-trees in the system. After construction of the upper layer R-tree, whenever a new Location Update Message (LUM) is received from a moving object (for example from a GPS associated with a vehicle), the lower R-tree is created or updated for the leaf node of the upper R-tree corresponding to the location of the update. Suppose we have a dataset containing information about Dallas-Fort Worth Road Network, to track movement of vehicles in this metroplex. By considering the aforementioned access structure, the upper layer R-tree indexes the Dallas-Fort Worth road network, and the lower layer R-trees store vehicles' movements. Given a query such as "Retrieve all vehicles that moved into Summit Avenue from 9:00am to 9:10am", according to the upper level R-tree, we find the leaf node corresponding to Summit Avenue, and with the aid of lower R-tree, we can locate the vehicles that passed into this avenue in that specific time interval ([9:00, 9:10]). The performance of the FNR-tree (as the earliest effort in Network-Constrained R-tree) was studied in [12]. It compared this access method

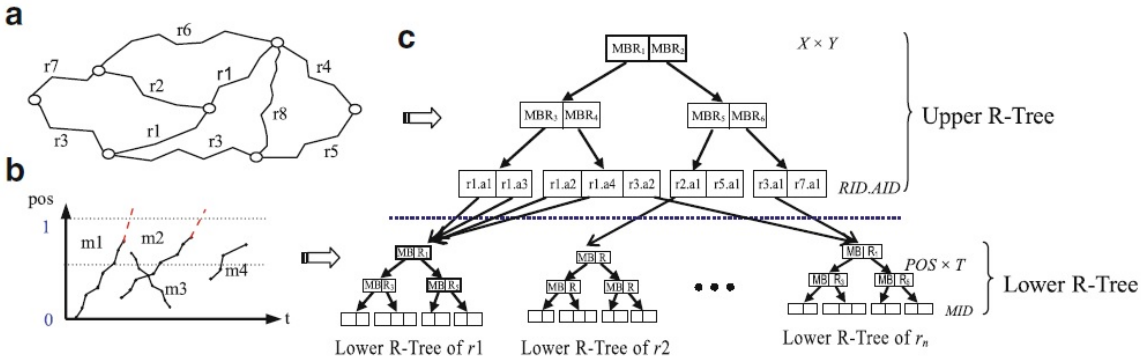

Figure 3.2: Structure of UTR-tree. (a) Traffic Network. (b) T-Units submitted in route r1. (c) The corresponding UTR-Tree

with the 3D R-tree under various datasets [7] and queries. Almost in all cases the FNR-tree significantly outperforms the 3D R-tree. The FNR-tree had better space utilization and smaller data size per moving object. Also, it supports range queries much more efficiently.

Although Network-Constrained R-trees are really efficient and they seem to be ideal for fleet management applications, they have a drawback when answering some kinds of queries. Since in Network-Constrained indexing methods the upper-layer R-tree represents Network data, queries that are not including spatial restriction (purely temporal queries, id queries, and path queries) may not be answered efficiently. In other words, in queries that are not spatial restricted, all records are read, and the existing index does not help. For instance, given the aforesaid indexing structure for Dallas-Fort worth including road network and moving objects' status, a possible query would be: "Return the location of moving object *mid* at time t_j". This "simple query" system requires a search of all records to find the result.

To overcome this shortcoming, we propose a new indexing method, Temporal Enhanced Network-Constrained R-tree (TENC R-tree), that improves the exist-

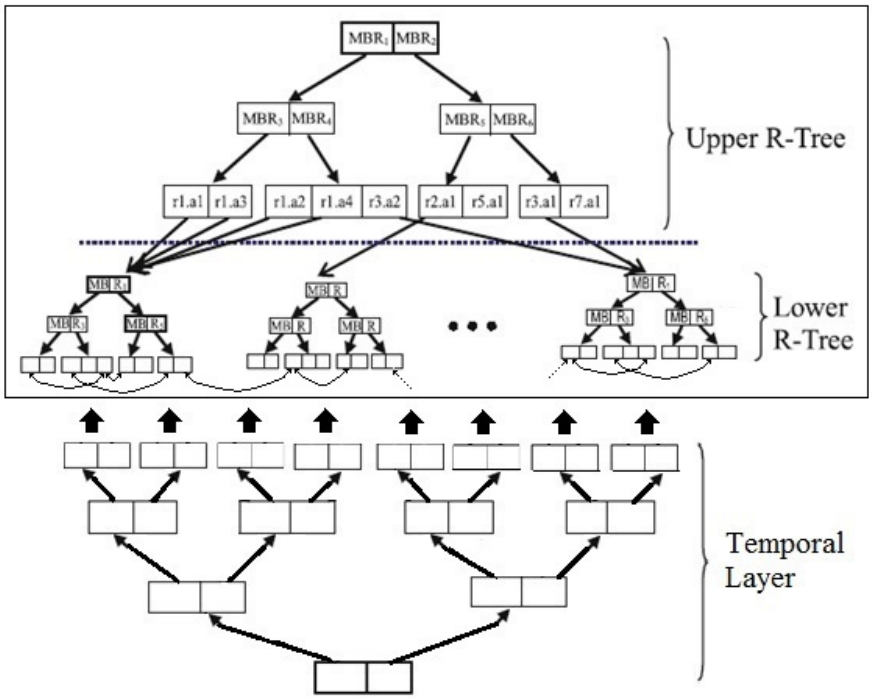

Figure 3.3: Structure of TENC-Rtree

ing Network-Constrained access methods to support spatial, temporal, and spatio-temporal queries, as well as id queries and path queries.

3.3 Structure model

Our idea can be exploited to enhance *all* Network-Constrained moving object indexing methods, but we describe it within the framework of UTR-tree [25, 26, 8] as the base structure for our idea. The UTR-tree is a hybrid structure (similar to other Network-Constrained data structures). The upper R-tree is edge-based by focusing on road segments (edges) between road intersection points, resulting in smaller granularity (than route-based). Therefore the intersections between different MBRs are significantly reduced. The lower layer is a

forest of route-based R-trees (greater granularity) with time interval and location, so the location update and index maintenance can be reduced.

The upper R-tree leaf nodes contain entries of the form $\langle MBR_{xy}, rid.eid, pt_{route}, pt_{tree}\rangle$ where MBR_{xy} is the Minimum Bounding Rectangle (in the $X \times Y$ plane) of edges, *rid.eid* is a combination of the route and edge identifier, pt_{route} is a pointer to the detailed route record, pt_{tree} is a pointer to the lower R-tree corresponding to the route *rid*. The root or interval nodes contain entries of the form $\langle MBR_{xy}, pt_{node}\rangle$ when MBR_{xy} is the MBR containing all MBRs of its child records, and pt_{node} is a pointer to the child node.

The lower part of the UTR-tree contains a forest of R-trees such that each R-tree indexes the trajectories that were received from moving objects in a certain route (e.g. Summit Ave.). The leaf nodes contain entries of the form $\langle MBR_{pt}, mid, mv_s, mv_e\rangle$ where MBR_{pt} is the Minimum Bounding Rectangle of the associated trajectory units in $POS \times T$ plane (POS is a value in [0, 1], which shows the position of a moving object in a route, normalized to a value between 0 and 1), *mid* is the identifier of the moving object, and mv_s and mv_e are two consecutive motion vectors that form the trajectory units. A motion vector is a snapshot of a moving object at a certain time instant and it is defined as follows:

$$mv = (t, rid, pos, \overline{v}) \tag{3.1}$$

UTR-tree supports queries on current and future positions of moving objects. Allocating a value to mv_e and an MBR for current status needs more computational effort and it is calculated then stored. We do not consider the calculations in this work since we just intend to study the basics of this structure for proposing our idea, see [8] for calculation details.

Figure 3.2 sheds light to the structure for UTR-tree [6, 7, 8]. Figure 3.2(a) shows the Traffic Network. Some part of this Network is indexed in the upper R-tree in Figure 3.2(c). Some submitted update trajectory units for route r1 are shown Figure 3.2(b). The corresponding R-trees for these trajectory units are indexed in the Lower R-trees in Figure 3.2(c).

3.4 THE TENC R-TREE

In this section, we propose a new index structure to answer spatial, temporal, and spatio-temporal queries, as well as path queries (Strict-path query and Plain-Path query) efficiently. This structure is called **Temporally Enhanced Network-Constrained R-tree** (TENC R-tree), and solves the shortcomings in the other Network-Constrained access methods. Most recently developed Network-Constrained access methods can store and retrieve the state of moving objects according to spatial aspects of data then on temporal aspects restricted to a specific road segment. Therefore for (purely) temporal queries (e.g. retrieve locations of all cabs at time t_j) or query based on moving object identifier, ID-based query (e.g. retrieve the trajectories of moving object *mid* from t_s to t_e), the system needs to search all records to retrieve the result. Moreover, these indexing methods cannot help in the case of Strict-Path queries. This query returns all the trajectories that strictly follow a given path,π, within a temporal period defined by t_s and t_e. In the following, subsection 3.4.1 presents the TENC R-tree index structure, and 3.4.2 and 3.4.3 present insertion and search algorithms, respectively.

3.4.1 Index Structure

The index proposed in this work is based on the UTR-tree [6, 7, 8] structure that is associated to edges for the Network model. Our idea can be exploited to enhance

all Network-Constrained moving object indexing methods, but we describe it within the framework of UTR-tree. In the UTR-tree, the lower part is a forest of R-trees that indexes the trajectory units. Since this structure is based on the spatial aspect of data, it cannot answer (purely) temporal queries and ID-based queries efficiently. Also, it is not be able to put up results for Strict-path queries efficiently. To overcome these shortcomings, we add another R-tree that indexes leaf nodes of the lower level R-tree according to t_s and t_e corresponding to time instants in mv_s and mv_e (in the lower R-tree), and the moving object identifier *mid*. This R-tree, which we call the *Temporal layer*, is shown in Figure 3.3. Moreover, in order to be able to support path query types (Strict and Plain), we added pointers to the leaf entries of lower R-tree. These pointers link each entry to the previous and next entries of the same trajectory. Through these links, we can move on the trajectory of a moving object. These pointers are called *trajectory pointers*. In Figure 3.3, the structure in the box is a modified UTR-tree (UTR-tree with *trajectory pointers*). It is worth mentioning that our Temporal Layer and trajectory pointers are independent of this specific structure (UTR-tree) and can be applied on other Network-Constrained access methods such as FNR-tree, MON-tree.

The temporal layer is an R-tree with some modifications. It is built on time period and moving object identifier in order to process the (purely) temporal and ID-based queries. Therefore, the leaf nodes are in the form of $(\langle MBR_{mid,t}, mid, mv_s, mv_e, prev, next\rangle)$, where $MBR_{mid,t}$ is the Minimum Bounding Rectangle of the associated trajectory units in $MOID \times T$ plane, *mid* is the identifier of the moving object, and mv_s and mv_e are two consecutive motion vectors that form the trajectory units (see Equation 3.1). *prev* and *next* are pointers to the previous and next entry of the trajectory. The root and internal nodes contain entries of the form $\langle MBR_{mid,t}, pt_{node}\rangle$. For immediate parents of leaf nodes, $MBR_{mid,t}$ is the

MBR containing all time points of moving object *mid*'s motion vectors in (t_s and t_e), and for other internal nodes and root, $MBR_{mid,t}$ is the MBR containing all MBRs of its child records. pt_{node} is a pointer to the child node.

3.4.2 Insertion

In this section, we consider movement insertion in TENC R-tree. Movement insertion happens whenever a Location Update Message (LUM) is received (i.e. a moving object changes its speed, direction, or the route) [12]. The movement insertion takes LUM that contains either one motion vector (speed-change), or three motion vectors ($\langle mv_{a1}, mv_{a2}, mv_{a3} \rangle$, where mv_{a1} and mv_{a2} correspond to the junction location and mv_{a3} corresponds to the location when a route-change location update is triggered) as an argument. The movement insertion algorithm for TENC R-tree is given in Algorithm 1. This algorithm is an extension to the Insertion Algorithm for UTR-tree [6, 7, 8]. We focus on insertion over the extended part and skip the Insertion Algorithm for UTR-tree.

When an LUM is received, the system checks the size of LUM. In the cases that the LUM has one motion vector, mv_a, the algorithm retrieves the last motion vector for *mid* that has sent the LUM (mv_n). If there does not exist any record in the database for *mid*, a new $RTree_{Temporal}$ entry is built and intialized by $\mu(mv_a)$, and two *NULL* values, $\emptyset$, for *prev* and *next* pointers (procedure *Create-Index-Entry*). Then, this new entry is inserted into $RTree_{Temporal}$. However, if mv_n exists for *mid*, after building a new $RTree_{Temporal}$ entry, the algorithm updates the motion vector unit related to mv_n as well as *prev* and *next* pointers.

In the cases that the LUM has three motion vectors, mv_{a1}, mv_{a2}, and mv_{a3}, the flow of building and inserting of the $RTree_{Temporal}$ entries is similar to one motion vector LUM, however we are involved with three motion vectors.

Algorithm 1 Movement INSERTION

1: **procedure** INSERT(LUM)
2: **while** MOD is running **do**
3: Receive LUM from moving object mid;
4: **if** LUM contains 1 motion vector $mva = (t_a, rid_a, pos_a, \overline{v_a})$ **then**
5: Let mv_n be the current active motion vector for moving object mid;
6: **if** mv_n is null **then**
7: Insert trajectory $\mu(mv_a)$ for mid to the Lower R-tree according to the *UTR-tree*;
8: Create-Index-Entry($RTree_{Temporal}, mid, \mu(mv_a), \emptyset, \emptyset$);
9: Propagate($RTree_{Temporal}, mid, \mu(mv_a)$);
10: **else**
11: Update $\mu(mv_n, mv_a)$ for mid in the Lower R-tree according to the *UTR-tree*;
12: Create-Index-Entry($RTree_{Temporal}, mid, \mu(mv_a), \emptyset, \emptyset$);
13: Let $\mathcal{A}_1$ be the address of last entry for mid in $RTree_{Temporal}$;
14: Let $\mathcal{A}_2$ be the address new built entry for mid in $RTree_{Temporal}$;
15: $\mathcal{A}_1.next = \mathcal{A}_2$
16: $\mathcal{A}_2.prev = \mathcal{A}_1$
17: Propagate($RTree_{Temporal}, mid, \mu(mv_n, mv_a)$);
18: Propagate($RTree_{Temporal}, mid, \mu(mv_a)$);
19: **else**
20: **if** LUM contains 3 motion vectors $mv_{a1}, mv_{a2}, mv_{a3}$ **then**
21: Let mv_n be the current active motion vector for moving object mid;
22: Update $\mu(mv_n, mv_{a1})$ and $\mu(mv_{a2}, mv_{a3})$ for mid in the Lower R-tree according to the *UTR-tree*;
23: Let $\mathcal{A}_1$ be the address of last entry for mid in $RTree_{Temporal}$;
24: Propagate($RTree_{Temporal}, mid, \mu(mv_n, mv_{a1}),$);
25: Create-Index-Entry($RTree_{Temporal}, mid, \mu(mv_{a2}, mv_{a3}), \emptyset, \emptyset$);
26: Propagate($RTree_{Temporal}, mid, \mu(mv_{a2}, mv_{a3})$);
27: Let $\mathcal{A}_2$ be the address of last entry for mid in $RTree_{Temporal}$;
28: Create-Index-Entry($RTree_{Temporal}, mid, \mu(mv_{a3}), \emptyset, \emptyset$);
29: Let $\mathcal{A}_3$ be the address new built entry for mid in $RTree_{Temporal}$;
30: $\mathcal{A}_1.next = \mathcal{A}_2$
31: $\mathcal{A}_2.prev = \mathcal{A}_1$
32: $\mathcal{A}_2.next = \mathcal{A}_3$
33: $\mathcal{A}_3.prev = \mathcal{A}_2$
34: Propagate($RTree_{Temporal}, mid, \mu(mv_{a3})$);

3.4.3 Search

As we mentioned before, TENC R-tree is able to answer spatial ($I_x \times I_y$), temporal (I_t), spatio-temporal ($I_x \times I_y \times I_t$) and ID queries effienctly (I_x, I_y, and I_t are intervals in X, Y, and T domains). Furthuremore, TENC R-tree supports path queries.

In the case of spatial, temporal, and spatio-temporal queries, the input of the query is a range in X × Y × T. When processing a spatial or spatio-temporal query, the system queries the upper R-tree, and receives a set of (*rid*, *pos*), then for each (*rid*, *pos*) the corresponding lower R-tree is searched to find the trajectory units intersecting with the query, as in the UTR-tree.

However, when processing a ID query and/or a pure temporal query (I_t), the system queries the Temporal layer, and receives a set of tuples containing (*rid*, *pos*, *mid*). Thus, with the help of *rid* and *pos*, we will be able to return the locations corresponding to moving object(s) based on the *rid* (the route) and *pos* (exact location in the route rid). The search algorithm for TENC R-tree is given in Algorithm 2.

Algorithm 2 SEARCH

```
1: function SEARCH(I_x, I_y, I_t, mid, R)
2:     Result= {};
3:     if query does not contain Ix × Iy then
4:         Search the Temporal layer R-Tree according to I_t × mid, and receive a set of n tuples {T_1, T_2, ..., T_n} ;
5:         for i =1 to n do
6:             if (T_i) satisfies R then
7:                 Result= Result ∪(T_i);
8:     else
9:         Search the upper R-Tree according to I_x × I_y, and receive a set of n pairs (rid_i, pos_i);
10:        for i =1 to n do
11:            for ρ ∈ pos_i × I_t do
12:                μ ⟵ the set of Tuples whose trajectory units are in RTree_low(rid_i) which intersect ρ;
13:                for T in μ do
14:                    if T.mid = mid and T satisfies R then
15:                        Result= Result ∪ T
    return Result
```

In the case of Stict-path query, the input of the query is a *path*, π, and a time period [14]. This query returns all moving objects, *mid*s, along with their (sub)trajectories that strictly follow the path,π, within a temporal period defined by t_s and t_e. In other words, the result of this query is set of *mid*s plus (sub)trajectories that have π as their subtrajectory, enters π at a time point equal to or later than t_s, and leaves π at a time point equal to or before t_e [14]. There is another path query called Plain-path query [33]. This query returns the set of moving objects and trajectories which visit at least one edge in the query path, π within the specific time period. Since Strict-path query is more strict in the evaluation of both the spatial and temporal components, in this work, we come up with a search algorithm for Strict-path query.

Algorithm 3 Strict-path query SEARCH

```
1: function SPQSearch(π, t_s, t_e)
2:     Result= {};
3:     Let SP = start(π);
4:     T = SEARCH(SP, t_s, t_e);
5:     for t in T do
6:         while t.pos ==SP and t.time ¡t_e do
7:             t = t.next;
8:             SP = Next(π, SP)
9:         if t reached end(π) then Result = Result ∪ t;
       return Result
```

Table 3.1: Comprehensive Spatio-temporal queriesn

Query type	Query sub-type	Condition	Result
Coordinate-based query	Window query	Spatial region and Temporal Interval	1) mids + sub-trajectories 2) mids
	Temporal point query	Spatial point and Temporal Interval	1) mids 2) mids + Temporal point
	Time-slice query	Spatial region and Temporal point	1) mids 2) mids + Spatial point
	Point query	Spatial point and Temporal point	mid
	Pure Spacial query	Spatial point	1) mids 2) mids + Temporal point
		Spatial region	1) mids + sub-trajectories 2) mids
	Pure Temporal query	Temporal point	mids + Spatial point
		Temporal interval	mids + sub-trajectories
	k Nearest-neighbor qery	Spatial point and Temporal point	k mids
ID-based Query	-	ID and/or (Spatial region or point) and/or (Temporal interval or point)	1) sub-trajectories 2) Spatial point 3) Temporal point
Path-based Query	Plain	Path	mids + sub-trajectories
	Strict		

Algorithm 3 shows the Strict-path query search in TENC R-tree. The input is the path, π as well as t_s and t_e as the starting and ending time, respectively. The output of this query is set of (sub)trajectories that has π as its sub-trajectory, enters π at a time point equal to or later than t_s, and leaves π at a time point equal to or before t_e. It may be observed that path π is a sequence of spatial points that can be traveled on the fixed traffic network. In Algorithm 3, start(π) and end(π) return the first and the end spatial point of path, π. Next(π, SP) returns the immediate adjacent spatial point next to SP. In the first step, the algorithm retrieves all records that intersect start(π). Then, for each record t, it checks if the next point in the

Table 3.2: BerlinMod datasets characteristics

Category	Days	Vehicles	Trip	Moving points
1	2	141	1,797	346,657
2	6	447	15,045	2,998,674
3	13	894	62,510	12,091,785
4	28	2000	292,940	56,129,943

t's trajectory coincides with next point in π, and corresponds to time period $[t_s, t_e]$. The trajectories that have π as its sub-trajectory, enter π at a time point equal to or later than t_s, and leave π at a time point equal to or before t_e will be returned. It is worth mentioning that sub-trajectory checking is done by the help of *prev* and *next* pointers, which are one of the contributions of this work (see section 3.4.1).

3.5 EXPERIMENTAL EVALUATION

In order to examine the performance of the TENC R-tree, we compared it with UTR-tree as one of the most recent indexing method for network-constrained moving object databases. In order to do this, we implemented TENC R-tree based on UTR-tree by a C++ program. Our test environment was Win7 64-bit operating system, Intel Core i5 CPU 860 @ 2.80GHz, 12GB RAM. The dataset to evaluate the index structure comes from BerlinMod, a benchmark for spatio-temporal database management systems (STDBMS) by the University of Hagen [35]. The moving object data are sampled from simulated behavior of workers commuting between their homes and work places, and additional trips in their leisure time on the street network of the German capital Berlin with 3212 routes and 11449 edges [36]. This dataset consists of 4 different sizes of sub-datasets as well as their benchmark queries for them (see Table. 5.1). In the first step of our evaluation, we compare the index size between UTR-tree and our new Indexing method, TENC R-tree. In Figure 3.4 , we demonstrate the

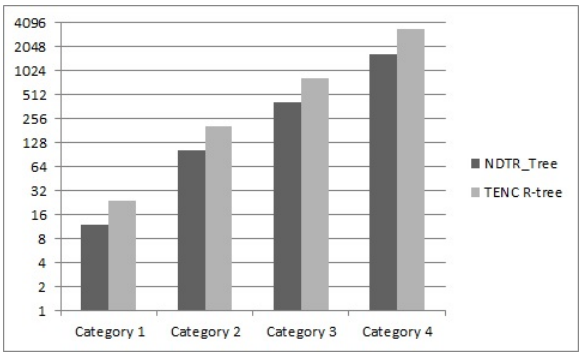

Figure 3.4: Index Size (MB)

TENC R-tree size compared to UTR-tree. As we can see the size of our indexing method is approximately twice as large as the UTR-tree, because of the additional temporal index.

In next step of our experiment, we compare 3 groups of queries:

- Pure Spatial and Spatio-tempral queries.

  In this group, the performance of UTR-tree and TENC R-tree is similar.

- Pure temporal and temporal ID queries.

  In this group TENC R-tree outperforms UTR-tree up to 20 times.

- Path query.

  In this group TENC R-tree outperforms UTR-tree up to 30 times.

To evaluate the UTR-tree and the TENC R-tree, we used queries, which are developed by [35] for each aforementioned dataset categories in Table 5.1. The Point and Region groups are pure Spatial queries containing 100 queries each. The Instance and Period groups are pure Temporal queries containing 100 queries each. The Spatio-temporal group, on the other hand, are obtained from a combination of the Region and Period group; therefore it contains 100 $\times$ 100=10000 queries. Furthermore, [35] supports 100 ID queries which can be combined with any other queries to build new

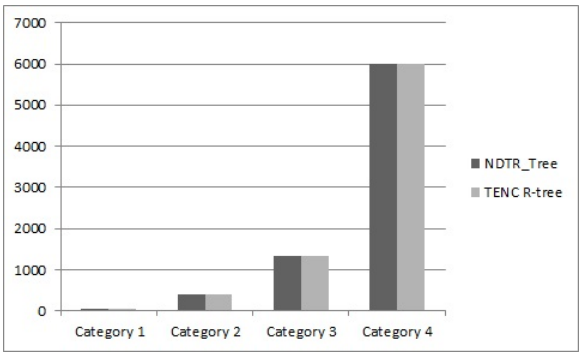

Figure 3.5: Average response time for a Region Query (ms)

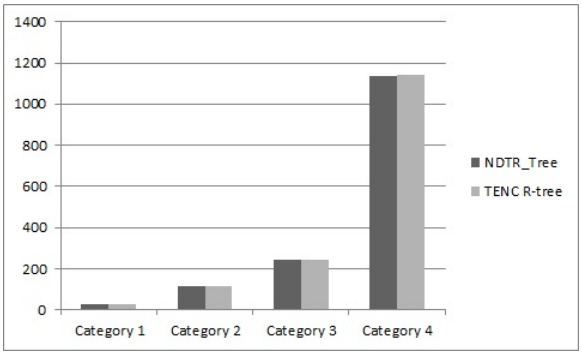

Figure 3.6: Average response time for a Window Query (ms)

complex queries. For instance, the ID queries are combined with temporal instance queries or temporal period queries in section 3.5.2.

BerlinMod [35] does not contain Path queries. Therefore, to do the Path Query evaluation, we built 100 different random paths. In order to do this, we chose 2 random points on the traffic network (of Berlin city) and found the shortest path between those points. We repeated the process 100 times to have 100 different paths. Then, we combined these 100 paths with the temporal queries of [35] to have 100 Path Queries. In subsection 3.5.3, we consider Path queries in more detail.

3.5.1 Spatial and Spatio-temporal Queries

In the first step of our experiment, we consider Spatial and Spatio-temporal queries. As we mentioned before, UTR-tree works efficiently for these types of queries. The proposed indexing structure in this work, TENC R-tree, uses the same procedure to deal with these kinds of queries. However, we compare UTR-tree and TENC R-tree to show they perform similarly. Considering table 3.1 there are two types of pure spatial query, Spatial region and Spatial point queries. Besides, there are 4 types of Spatio-temporal queries, window query, exact-point interval query, and point query. Out of these six different queries, we consider Spatial region query (Figure 3.5) and Window query (Figure 3.6), and we show UTR-tree and TENC R-tree have the same performance for these queries.

3.5.2 Pure Temporal and ID Queries

In this section, we present the first contribution of this work. As we discussed before, UTR-tree performs inefficiently when the search query contains no spatial features. In other words, in dealing with Pure temporal and ID queries, UTR-tree needs to scan the entire database to retrieve the result. However, since TENC R-tree has the additional layer, called Temporal layer (see section 3.4.1), it uses this layer to retrieve the result (see Algorithm 2).

As we can see in Figure 3.7 and Figure 3.8, the average response time for a pure temporal query can be up to 7 times faster in TENC R-tree than in UTR-tree. Further, this trend is more significant if we have Pure temporal ID query. Considering Figure 3.9 and Figure 3.10, we observe pure Temporal ID queries are up to 20 times faster in TENC R-tree than UTR-tree.

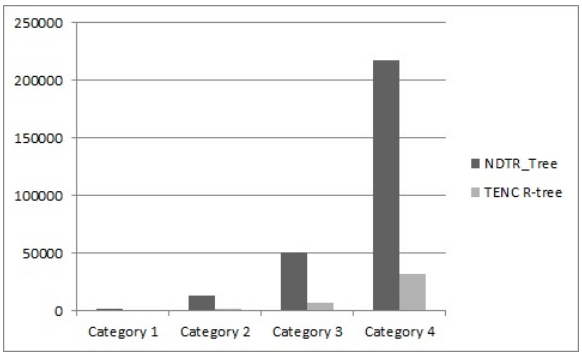

Figure 3.7: Average response time for a Time-slice Query (ms)

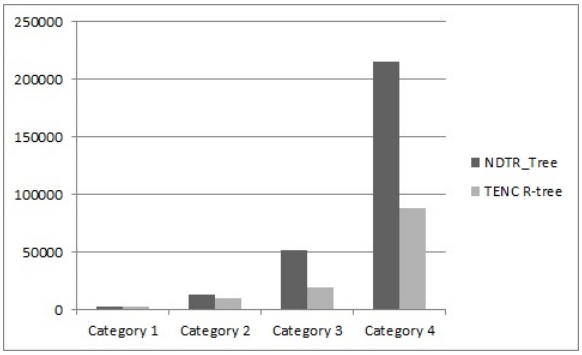

Figure 3.8: Average response time for a Temporal interval Query (ms)

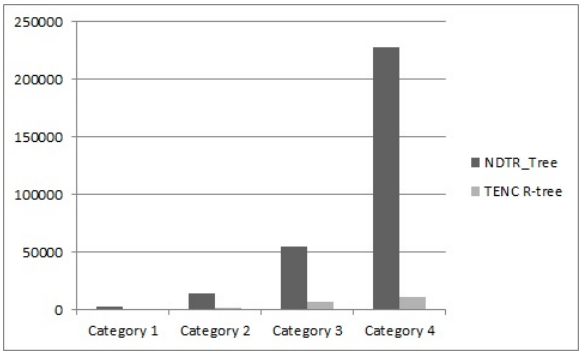

Figure 3.9: Average response time for a Space-slice ID Query (ms)

In the case of pure ID query (queries without any spatial or temporal feature), we can notice that TENC R-tree can outperform UTR-tree up to 8 times (see Figure 3.11).

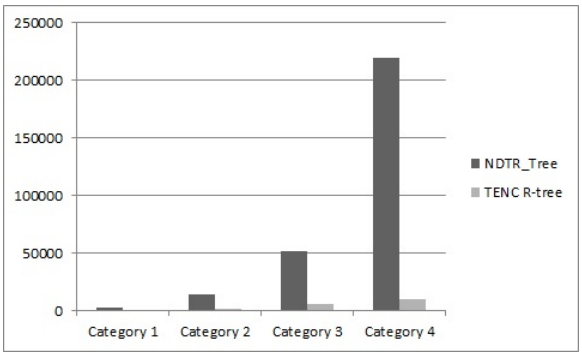

Figure 3.10: Average response time for a Temporal interval ID Query (ms)

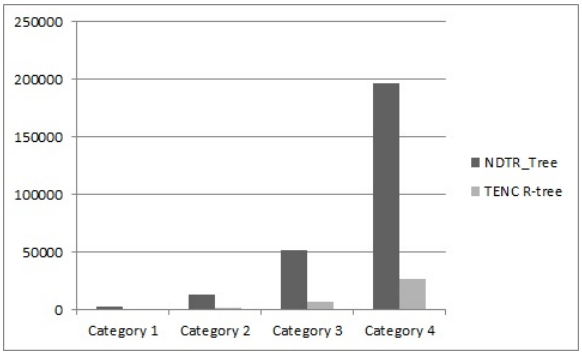

Figure 3.11: Average response time for a pure ID Query (ms)

3.5.3 Path Query

The next contribution of our indexing structure comes into play in the case of Path queries. In this section, we go over both plain (Figure 3.12) and strict (Figure 3.13) path queries. Thanks to *prev* and *next* pointers (see section Indexstructure), TENC R-tree can outperform UTR-tree up to 30 times. It is worth mentioning that, in the case of Strict path query, the performance of the SPQSearch() (Algorithm 3) is almost independent of dataset size. The reason behind this is once the algorithm finds the first leaf node in the tree, it just needs to follow the *next* pointers, and does not required to search through the tree anymore. (i.e. one spatio-temporal search, plus some pointer traversing).

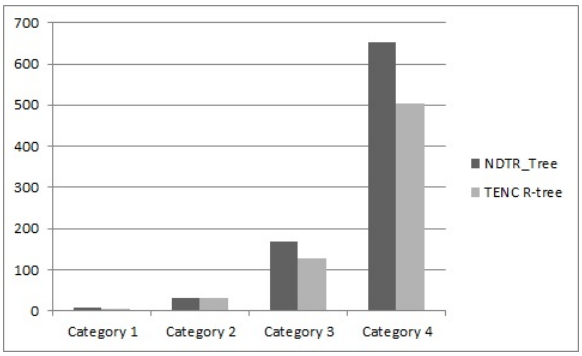

Figure 3.12: Average response time for a Plain Path Query (ms)

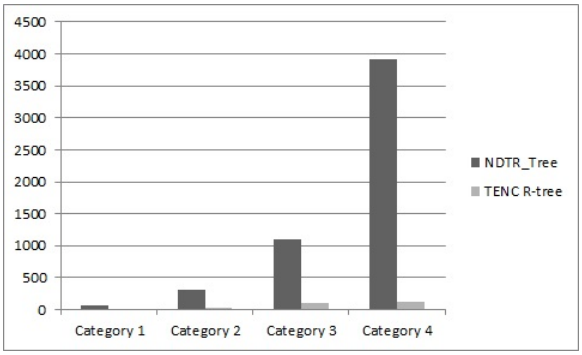

Figure 3.13: Average response time for a Strict Path Query ID Query (ms)

3.6 CONCLUSION

In this chapter, we proposed a new Spatio-Temporal indexing technique known as Temporally Enhanced Network-Constrained Enhanced-tree, which enhances the UTR-tree. We compared our indexing structure with UTR-tree for spatial queries, which include point and region queries, and for pure temporal queries which include instance and period queries. Our indexing structure, which was built to support pure temporal queries, ID queries, and Path queries gave significantly better performance (10 to 30x faster) when compared to the UTR-tree. We also considered spatio-temporal queries (both spatial and temporal) for which both of the indexing structure (UTR and TENC) showed similar performance

CHAPTER 4

Scalable Deep Traffic Flow Neural Networks for Urban Traffic Congestion Prediction

In this chapter, we try to predict the traffic flow of Traffic Network points, where we do not have any historical data about them, based on the traffic patterns of Traffic Network points. In the rest of this chapter, after having an introduction and covering the related work in Section 4.1, we start our work by some preliminaries in Section 5.5. We define all the main traffic flow concepts and then bring up the problem we are going to solve. We also, introduce two deep network model, and describe their structures broadly. In Section 4.3, we explain our models and methods in more details. Then, we experimentally evaluate our proposed prediction models and compare them with more simple models in section 4.4.

4.1 Introduction and related work

Traffic congestion leads to extra gas emissions and low transportation efficiency, and it wastes a lot of individuals' time and a hunge amount of fuel. Diagnosing congestion and building a pattern for predicting traffic congestion has been regarded as one the most important issues as it can lead to informal decisions on the routes that motorists take, and on expanding road networks and public transport. Research to predict traffic congested spots, especially in urban areas is thus very important.Typcally, congestion prediction can be used in Advanced Traffic Management Systems (ATMSs) and Advanced Traveller Information Systems in order to develope proactive traffic control strategies and real-time route guidance.[15]

In the last decades, concepts of traffic bottleneck and congestion propagation have been considered in many studies. Although most of these originate from Civil Engineering and Urban Transportation studies, the advent of super powerful computers and complex algorithms, traffic management and traffic flow prediction to become an interdisciplinary study.

In this regard, there have been various efforts to predict short-term traffic flow prediction, including mathematical equations [16, 17], simulation techniques [18], or statistical and regression approaches. However, traffic flow is based on individuals' decisions, which more likely can be modeled by Artificial Neural Network the best. In other words, traffic flows are made by individuals' decisions based on their knowledge about currenct traffic and their experiences about past traffic flows, which can be modeled by Artificial Neural Network. Using Neural Network for modeling traffic flow and congestion prediction came to the picture in 1993 in [19]. This work propose a network consisting of one *input* layer, one *hidden*, and one *output* layer. Although this structure was proven to perform well in many applications for predicting traffic flow and travel time and estimation, it was not efficient in lots of other, because of the simple structure. Therefore, some research uses a Neural Network, initially, to extract traffic flow patters (clustering), and then based on each pattern, they come up with a proper model to predict traffic flow [20, 21, 22]. In this trend, [15] different predictors have different performance for various particular time periods. In other words, each predictor can have a super performance only in a particular time period. Therefore, they combined several predictors together as module to have a better performance for longer time periods.

The data regarding Traffic Flow and Traffic Congestion are two instances of Spatio-temporal data. They embady a location (Spatial Feature) and a time (Temporal feature). Besides, as we already mentioned, traffic flow and traffic congestion

are based on human actions [23]. In [24], the authors propose a fully automatic deep model for human-action-based spatio-temporal data. This model first utilizes Convolutional Neural Network model (CNN) to learn the spatio-temporal features. Then, in the second part of this model, they use the output of the first step to train a recurrent neural network model (RNN) in order to classify the entire sequence. [24] does not mention traffic issues as one of the possible applications of their work, however it seems promising to make some model, which is inspired by their model, to predict traffic flow and congestion.

In 2015, [24] Deep Learning theory was put into practice for large-scale congestion prediction. To this end, they utilized Restricted Boltzmann Machine [41] and Recurrent Neural Network [42] to model and predict the traffic congestion. In order to do this, they convert all the speed data of Taxis in Ningbo, China to binary values (i.e. the speed more than a threshold is 1, otherwise it is 0), and then call these values *Congestion Conditions*. Therefore, the network congestion condition data will be a matrix as follows:

$$\begin{bmatrix} C_1^1 & C_1^2 & C_1^3 & \dots & C_1^T \\ C_2^1 & C_2^2 & C_2^3 & \dots & C_2^T \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_N^1 & C_N^2 & C_N^3 & \dots & C_N^T \end{bmatrix}$$

Each element in the matrix indicates congestion condition in a specific point at a specific time slot. Therefore, C_n^t represents the congestion condition on the nth point of the traffic network at tth time slot (The Network has N point). Give this matrix

to the model presented in [24], the result will be the predicted traffic condition for each point at *T+1*.

$$\begin{bmatrix} C_1^1 & C_1^2 & C_1^3 & \dots & C_1^T \\ C_2^1 & C_2^2 & C_2^3 & \dots & C_2^T \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_N^1 & C_N^2 & C_N^3 & \dots & C_N^T \end{bmatrix} ======\Rightarrow \begin{bmatrix} C_1^{T+1} \\ C_2^{T+1} \\ \vdots \\ C_N^{T+1} \end{bmatrix}$$

Although [24] presented a good performance for predicting traffic condition, it has some drawbacks:

- The traffic condition is limited in either *Congested* or *Not-Congested* (1 or 0). However, in real applications, we usually need a *range* of values (or colors in case of Map) to show amount of traffic flow.
- The traffic condition is set based on a specific threshold (for example 20 km/h). If the average speed is less than the threshold the traffic condition will be set as *congested*, otherwise it will be *Not-congested*. Nevertheless, having a specific threshold for the whole network is inappropriate. Rather, the traffic condition is supposed to be set based on the ratio of average speed of vehicles to possible max speed (Speed limit).
- In the model presented in [24], authors did not consider any order for Network points as the input (the rows of the matrix). However, the spatial influence of adjacent network points should be taken into consideration.

In this work, we try to predict the traffic flow of Traffic Network points, where we do not have any historical data about them, based on the traffic patterns of Traffic Network points. Therefore, our contributions are as follows:

1. We formally define the traffic flows prediction concepts.

2. We introduce a normalized data representation, which can be used in Neural Network algorithms, or other methods.
3. We present a Deep Convolutional Network, which can be able to learn traffic flow of different traffic points.
4. Then, we present a Recurrent Neural Network, which, apart from its structure, can do the same as Convolutional Network.
5. Both of these models are able to predict n-level traffic prediction for different points of the traffic (e.g. Quiet, light traffic, heavy traffic, congested, etc.)
6. They also put up the predicted average speeds on different points of the traffic network based on the speed limits in that point (e.g. 0.65 of speed limit).

4.2 Preliminaries

In this section, we give formal definitions of traffic flows prediction problem in subsection 4.2.1. Then in subsection 4.2.2, we formally introduce the problem presented in this work.

4.2.1 Formal Definitions

A traffic Network comprises a set of roads, as well as set of junctions. Junctions can be intersections of streets, exit-entrance of highways, roundabouts, beginning-end point of a road, U-turns, etc. In the subject of traffic flows, we can consider junctions as the main points of the network, because these points can be major factors of changing the traffic flows. By way of explanation, typically a traffic flow may not change significantly between two junctions, but it may change because of traffic lights, exit-entrance, and so on. Consequently, almost all of the traffic Network points and traffic sensors are installed in junctions.

Definition 1 (Network Points). In this study, we represent a road (streets, highways, etc.) by N points based on the junctions on that road. Each of these points may indicate the traffic condition (for example congested) on that point. Consequently, the whole Traffic Network Condition can be presented by set of all junctions on that Network, which are called *Network Points* and denoted by $\mathcal{N}$. It is worth mentioning that each Network point has spatial interaction with adjacent Network points, and traffic flow conditions of a point may get/have influence from/on the adjacent points.

Definition 2 (In-flow sequence). Assume $\mathcal{S}$ is a Network point and L_1, L_2, ..., and L_n are adjacent points on the traffic network, which have flow to $\mathcal{S}$, such that L_1 is the closest point to $\mathcal{S}$, and L_n is the farthest. The In-flow sequence of $\mathcal{S}$ is denoted as $\text{In}(\mathcal{S})$.

$$\text{In}(\mathcal{S}) : L_n \rightarrow L_{n-1} \rightarrow ... \rightarrow L_2 \rightarrow L_1 \rightarrow \mathcal{S}$$

Definition 3 (Out-flow sequence). Assume $\mathcal{S}$ is a Network point and R_1, R_2, ..., and R_m are adjacent points on the traffic network, which $\mathcal{S}$ has flow to them, such that R_1 is the closest point to $\mathcal{S}$, and R_m is the farthest. The Out-flow sequence of $\mathcal{S}$ are denoted as $\text{Out}(\mathcal{S})$.

$$\text{Out}(\mathcal{S}) : \mathcal{S} \rightarrow R_1 \rightarrow R_2 \rightarrow ... \rightarrow R_{m-1} \rightarrow R_m$$

Definition 4 (Point snapshot). Assume $\mathcal{S}$ is a Network point and L_1, L_2, ..., and L_n are $\text{In}(\mathcal{S})$, and R_1, R_2, ..., and R_m are $\text{Out}(\mathcal{S})$, such that:

$$L_n \rightarrow ... \rightarrow L_2 \rightarrow L_1 \rightarrow \mathcal{S} \rightarrow R_1 \rightarrow R_2 \rightarrow ... \rightarrow R_m$$

Point snapshot at time t, denoted as $\text{Snapshot}(\mathcal{S}, t)$, is the *Traffic Condition* of [L_n, ... , L_2, L_2, $\mathcal{S}$, R_1, R_2, ..., R_m] at time series of [t-δ, ..., t-*1*, t]. This time series indicates a sequence of the last δ time points with a specific time interval between

each two consecutive time points (for instance, 20 minutes) . Formally, Snapshot($\mathcal{S}$,t) is defined as follows:

$$
\begin{array}{c}
 \\
L_n \\
 \\
L_{n-1} \\
\vdots \\
L_1 \\
 \\
\mathcal{S} \\
 \\
R_1 \\
\vdots \\
R_{m-1} \\
 \\
R_m
\end{array}
\begin{array}{c}
\begin{array}{cccc} t-\delta & ... & t-1 & t \end{array} \\
\left[\begin{array}{cccc}
c_{t-\delta}^{L_n} & \dots & c_{t-1}^{L_n} & c_t^{L_n} \\
c_{t-\delta}^{L_{n-1}} & \dots & c_{t-1}^{L_{n-1}} & c_t^{L_{n-1}} \\
\vdots & \ddots & \vdots & \vdots \\
c_{t-\delta}^{L_1} & \dots & c_{t-1}^{L_1} & c_t^{L_1} \\
c_{t-\delta}^{\mathcal{S}} & \dots & c_{t-1}^{\mathcal{S}} & c_t^{\mathcal{S}} \\
c_{t-\delta}^{R_1} & \dots & c_{t-1}^{R_1} & c_t^{R_1} \\
\vdots & \vdots & \ddots & \vdots \\
c_{t-\delta}^{R_{m-1}} & \dots & c_{t-1}^{R_{m-1}} & c_t^{R_{m-1}} \\
c_{t-\delta}^{R_m} & \dots & c_{t-1}^{R_m} & c_t^{R_m}
\end{array}\right]
\end{array}
$$

Where c_τ^σ indicates *traffic condition* of point σ at time τ. *Traffic condition* is a value between 0 and 1 ($0 \leq c_\tau^\sigma \leq 1$), and shows the ratio of the average speed of vehicles to the speed limit in point σ at time τ. Although it is extremely rare that the average speed exceeds speed limit, in cases which exceed, c_τ^σ is considered as 1. It is worth mentioning that Snapshot($\mathcal{S}$,t) is the input unit for predicting the traffic flow of $\mathcal{S}$ at time *t+1*(for example in 20 minutes). In other words, for predicting the traffic flow of $\mathcal{S}$ at time *t+1*, we need recent δ traffic condition of point $\mathcal{S}$, as well as recent δ traffic condition of In($\mathcal{S}$), and recent δ traffic condition of Out($\mathcal{S}$).

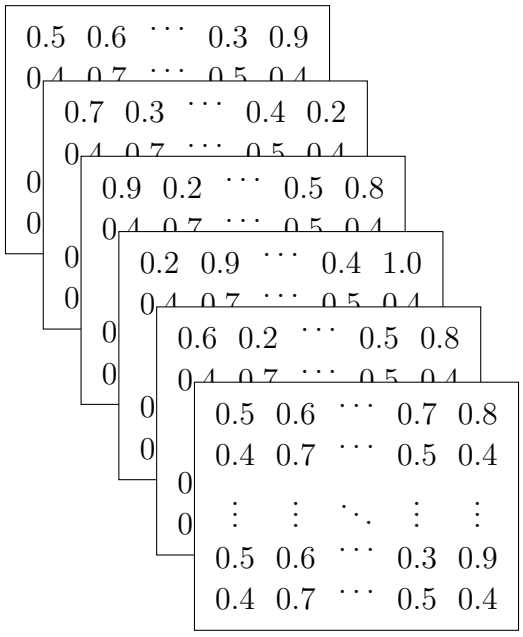

Figure 4.1: Schematic view of SNAPSHOT($\mathcal{N}$, t)

Definition 5 (Network snapshot). Assume $\mathcal{N} = \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, ..., \mathcal{S}_N\}$. Snapshot of $\mathcal{N}$ at time t, denoted by SNAPSHOT($\mathcal{N}$, t), and defined as follows:

$$\text{SNAPSHOT}(\mathcal{N},\, t) = \bigcup_{i=1}^{N} \text{Snapshot}(\mathcal{S}_i, t)$$

Where Snapshot($\mathcal{S}_i, t$) is the Point snapshot of $\mathcal{S}_i$ at time t, and $\bigcup$ is the union of snapshots of the Network points. Therefore, Snapshot($\mathcal{S}_i, t$) is a set of all point snapshots of Network points. Fig.4.1 schematically present SNAPSHOT($\mathcal{N}$, t).

In Fig.4.1, each box consists of the snapshot of one Network point at time t, thus all boxes indicate snapshot of the whole traffic network. Assume, Fig. 4.1 is the current snapshot of the traffic network (SNAPSHOT($\mathcal{N}$, *now*)). And, It may be the input for prediction of the next time point (for example, in 20 minutes).

4.2.2 Problem Definition

Assume we have the historical Network snapshots of a region (e.g. North California), gained from traffic detectors located on Traffic points (Definition 1). Our

system can learn traffic patterns from this historical data. Suppose, we have the current and the recent Network snapshots of another region (e.g. Southern California), because recently the latter region was equipped with traffic sensors. In this work, we try to predict the traffic flows of the latter region based on the traffic patterns learned from the former traffic network.

Problem. Given the historical traffic observations of Network $\mathcal{N}_1$ and Snapshot($\mathcal{S}$, *now*), $\mathcal{S} \notin \mathcal{N}_1$, predict traffic condition of $\mathcal{S}$ at $(now + 1)$, where $(now + 1)$ is the next time point (e.g. traffic condition in 20 minutes from now).

4.3 Deep traffic flow network

In this section, we try to use two deep learning model to solve the problem defined in subsection 4.2.2. In order to do this, we utilize two prominent algorithms, namely, *Convolutional Neural Network* [43] in subsection 4.3.1 and Long Short-Term Memory [44] in 4.3.2.

4.3.1 Deep Traffic Flow convolutional Network

Now that we defined all the main concepts about Traffic Flow Prediction, we need to introduce our Deep Traffic Flow convolutional Network. Fig.4.2 illustrates the model while training. As seen, the inputs to this model are categorized in two broad groups, namely, *Traffic Condition* and *Incidents*. *Traffic Conditions* are the set of Network snapshots (See Definition 5). In other words, *Traffic Conditions* is as follows:

$$\textit{Traffic Conditions} = \bigcup_{i=1}^{Z} \text{SNAPSHOT}(\mathcal{N}, t_i)$$

Where Z is the size of the dataset, chosen for training the model, and SNAPSHOT($\mathcal{N}, t_i$) is the Network snapshot in each training item. In Fig.4.2, we have N points in our

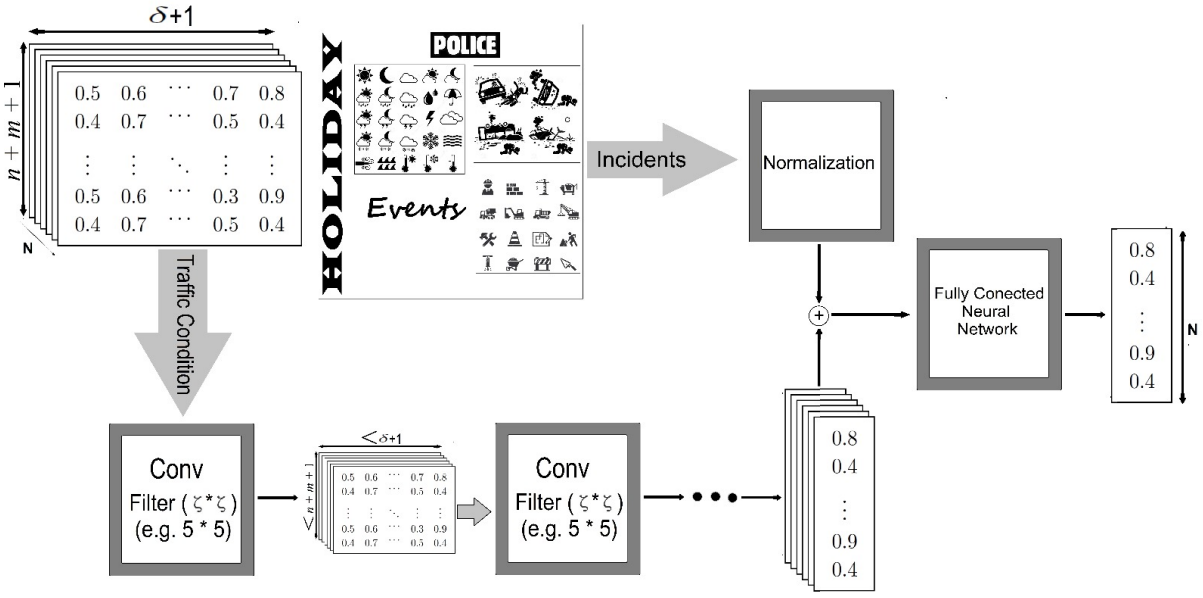

Figure 4.2: Deep Traffic Flow Convolutional Network

Network $\mathcal{N}$. For each Network point at a particular time point, we need to show the snapshot (a 2D array with size $((\delta + 1) \times (n + m + 1))$.

On the other hand, *Incidents* are Weather inputs or information about Car accidents, Holiday dates, Road construction, and Events, such as soccer match or concerts. Also, sometimes Police may change the traffic flow. Intuitively, we know these incidents may have huge influence on traffic flow. In the first step, past *Traffic Conditions* are given to the first layer of Convolutional Neural Network as the training set. Then, the result of first layer is given to the second Convolutional layer. This trend continues until we have a set of one dimensional arrays. As we know, in each layer of Convolutional, the size of input will decrease (because of the filter $(\zeta * \zeta)$ applied on the input).

When the output of a Convolutional layer is a set of 1D values, it is time to add the incident information to the output, and set the input for Fully-connected Network. The outcome Fully-connected Network outcome is the predicting values

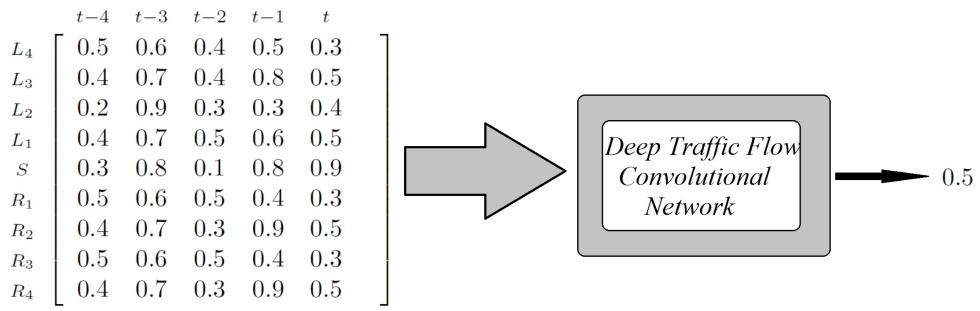

Figure 4.3: Test stage of Deep Traffic Flow Convolutional Network

of Traffic Flow at $t + 1$. While training, this output should be compared to the actual traffic flows, and *loss* value should be calculated. By getting help from *loss* value, the *weights* and *bias* values may be updated. In this work, we try to solve the problem introduced in subsection 4.2.2. In order to do this, we used the trained aforementioned model. Fig. 4.3 shows the test model of Fig. 4.2. Having the the traffic condition of point $\mathcal{S}$ (the point which we try to predict), and traffic condition of $\text{In}(\mathcal{S})$ and $\text{Out}(\mathcal{S})$ in time interval $[now - \delta, now]$, the output is the traffic condition of $\mathcal{S}$ at $now + 1$.

4.3.2 Long short-term memory Traffic Flow

In subsection 4.3.1, we introduce our Deep Traffic convolutional Network for solve the problem defined in section 4.2.2. In this subsection, we bring up another method, called *Long short-term memory Traffic Flow*. In this method, we use the Long short-term memory (LSTM) network [44]. LSTM is a Recurrent Neural Network [42] with more complex structure in the repeating modules. In other words, each repeating modules contains 4 interacting layers, thus LSTM avoids the long-term dependency problem. Fig. 4.4 shows an instance of an LSTM module.

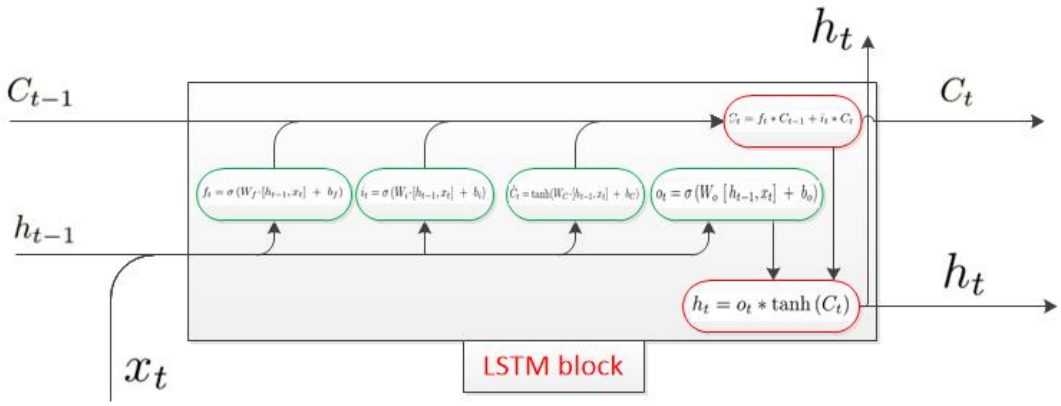

Figure 4.4: An instance of LSTM module

In this work, as we mentioned before, we try to predict the short-term future traffic condition for a Network point, where we do not have the historical Traffic condition about it, based on traffic patterns we learned from another road network. Considering the structure of LSTM, our whole network is as in Fig. 4.5. In Fig. 4.5, the traffic condition of point $\mathcal{S}$ (the point which we try to predict), and traffic condition of In($\mathcal{S}$) and Out($\mathcal{S}$) at time $t - \delta$, are given to the first LSTM module. Then, in the next step the traffic condition of point $\mathcal{S}$, In($\mathcal{S}$), and Out($\mathcal{S}$) at time $t-(\delta-1)$, are given to the second LSTM module. This trend continues until the last module. In the last module the output is the traffic condition of $\mathcal{S}$ at $now + 1$.

4.4 Experimental Evaluation

In order to examine the performance of our model, we explain our models in detail followed by introducing our dataset. In subsection 4.4.1, we describe our dataset, and in subsection 4.4.2, we bring up our method more in detail.

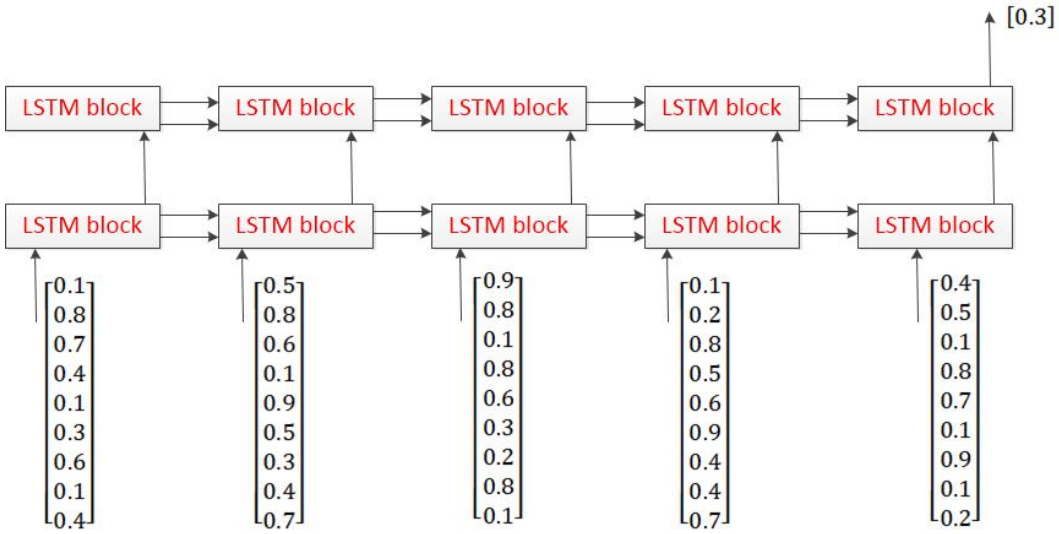

Figure 4.5: Long short-term memory Traffic Flow

4.4.1 Data set

In all experiments of this evaluation, we used real traffic data of California State. In order to do this, we used the data presented by Caltrans Performance Measurement System (PeMS)[45]. PeMS utilized over 39,000 detectors to collect real-time traffic data. These sensors cover the freeway system across all major metropolitan areas of the State of California. Therefore, PeMS prepared over 10 years of (historical) traffic flow data. The available data in PeMS are not limited to traffic flow data, but it also archived incidents data, such as, Car accidents, Weather information, Lane closures, etc. In this work, we trained our model for 51 locations on duration of 48 days. Then, we test the model for the same location for the next 12 days. The raw dataset and cleaned dataset are available in [46]. These datasets can be used as the benchmark for future Traffic Flow Prediction models.

The data in PeMS are raw data. In order to use them in our model, we did the following pre-processing steps:

4.4.1.1 Getting the data

The first step is getting the data from [45]. Thus, we download traffic flows (average speeds) of 58 consecutive locations of *US 101* highway for 60 days (each detector on the freeway indicates one location). The aforementioned 60-days is from "August 15, 2016" to "October 14, 2016". The granularity for the data is 5 minutes. By way of explanation, we have every-five-minute traffic flow of 58 Network points for 60 days (from "August 15, 2016" to "October 14, 2016").

4.4.1.2 Cleaning the data

The next step is the data cleaning. In order to do this, we checked the data and we noticed for a few temporal points, there is not any traffic flow information. For example, we have the traffic flow for Day_n at 13:45 and 13:55, but no data for 13:50. In this case, we calculate the mean values of data at 13:45 and 13:55, and consider them for 13:50.

4.4.1.3 Normalizing the data

The main part of our data gathering is the *normalizing* of the data. In our experiment, we consider the number of In-flow and Out-flow for each network point are 4 (see Definition 2 and Definition 3). The In-flow and Out-flow values can be defined based on condition of case study, such as, the distance between two consecutive Network Points, average speed, etc. Defining the size of In-flow and Out-flow can be consider as the future work. Also, we assume that in each point snapshot we have traffic conditions of time t, and 4 time steps before t ($\delta = 4$, see Definition 4). Therefore, point snapshots are 9×5 ($(n + m + 1) \times (\delta + 1)$). Although we 58 network position, we will not be able to make point snapshots for more than 50 of

them, because for the first 4 points we do not have In-flow, and the last 4 of them, we do not have Out-flow.

Besides, we batch the data in buckets of 30-minutes based on the day of the week and time. Put differently, all the traffic flows of Mondays at 8:00 am are considered in the same buckets. Thus, we assign values of "0 to 6" to the days of the week (i.e. 0 is assigned to Sunday, 1 is assigned to Monday, and so on). At same time we assign values "0 to 47" to the time of the day (i.e. 0 is assigned to [00:00, 00:30), 1 is assigned [00:30, 01:00), and so on).

Now that we convert time to two values, it is the time to normalize and prepare our data for *training* and the *test*. For normalizing the data, we need to convert all the values (traffic flow information and time) to values in the range [0, 1]. In order to do this, we divide all values by their possible maximum values. For this reason, we divide days and time points by 6 and 47 respectively. Also, we find the ratio of traffic flow of each point to the possible max speed (Speed limit). Hence, a sample of point snapshot is as follows:

0.5, 0.6

	$t-4$	$t-3$	$t-2$	$t-1$	t
L_4	0.5	0.6	0.4	0.5	0.3
L_3	0.4	0.7	0.4	0.8	0.5
L_2	0.2	0.9	0.3	0.3	0.4
L_1	0.4	0.7	0.5	0.6	0.5
S	0.3	0.8	0.1	0.8	0.9
R_1	0.5	0.6	0.5	0.4	0.3
R_2	0.4	0.7	0.3	0.9	0.5
R_3	0.5	0.6	0.5	0.4	0.3
R_4	0.4	0.7	0.3	0.9	0.5

In this instance, Day value and Time value are equal to 0.5 and 0.6, respectively (i.e. It is Wednesday at 14:00 to 14:30). In the matrix of traffic conditions, we have 9 rows, indicating the 4 In-flows, the 4 Out-flow, and the Source which is supposed to be predicting. Also, there are 5 columns, 1 column for time t, and 4 columns for the last 4 time points before t. The time interval between two consecutive time values is 5 minutes (the granularity we chose while getting the data).

4.4.2 Deep learning-based method

Now that we have the normalized cleaned data from the subsection 4.4.1, we need to implement two aforementioned methods in section 4.3, namely, *Deep Traffic Flow convolutional Network* and *Long short-term memory Traffic Flow* to learn the traffic flow (traffic pattern) of one specific Road Network, $\mathcal{N}_1$). Then, the learned models should be used to predict another road Network, $\mathcal{N}_2$ (see subsection 4.2.2).

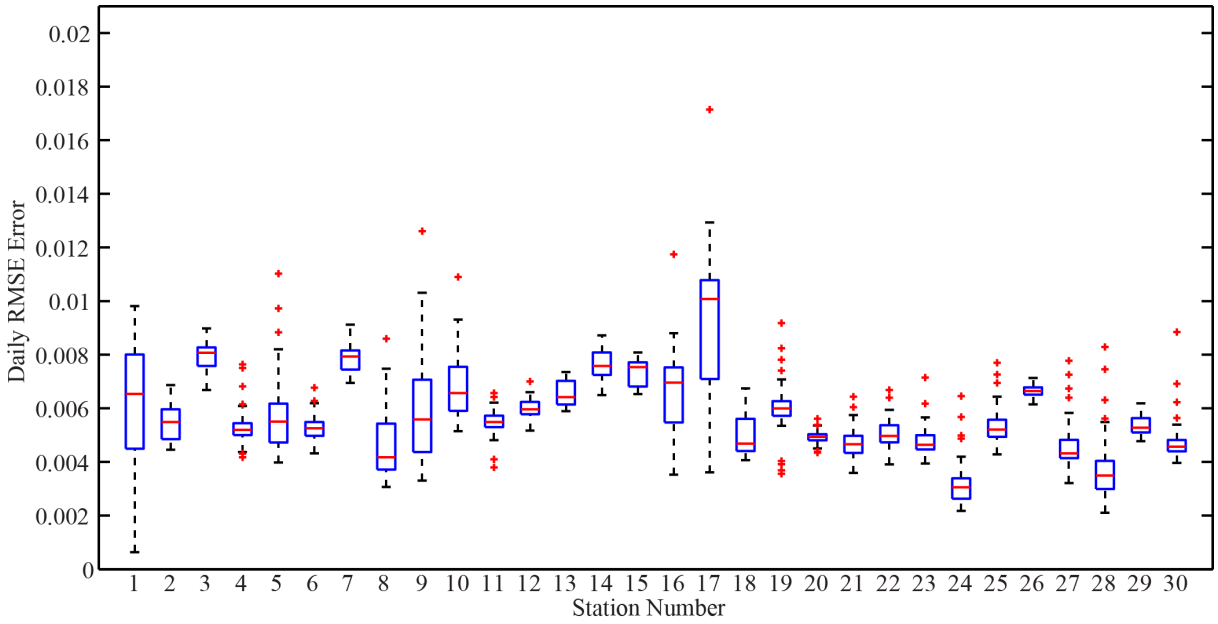

Figure 4.6: Daily RMSE Error for CNN

In this experiment, we use the first 20 Network points of normalized cleaned data as the training set and the rest 30 Network points as the test set. As we know, former Network points are southern part of dataset, and the latter ones belong to the northern section. Therefore, the two sets are disjoint in order to evaluate the generalization ability of the proposed methods (see subsection 4.2.2).

In order to evaluate the proposed methods, we employed a system as follows:

- Intel(R) Core(TM) i7 CPU 960 @ 3.20GHZ
- 8GB Ram
- Quadro 600 NVDIA GPU 1GB

We designed our Deep learning architectures using Lasagne Deep learning framework [47] installed on Theano [48].

The specifications of the proposed Networks are presented in Tables 4.1 and 4.2, where inputs and outputs size of each layer, as well as their filters size are tabulated.

Table 4.1: Specification of the proposed Convolutional Neural Network

Layer Type	**Input**	**Output**
Conv1	9×5	$7 \times 3 \times 64$
Conv2	$7 \times 3 \times 64$	$5 \times 1 \times 64$
Fully-Connected	320	32
Fully-Connected	32	1

Table 4.2: Specification of the proposed LSTM network

Layer Type	Input	Output	#Hidden Nodes
LSTM1	9×5	9×5	20
LSTM2	9×5	1	20

It is worth mentioning that, in this work, we used the Euclidean loss function to train the networks since our proposed problem is intrinsically categorized as Regression. Besides, intuitively, we know that traffic flow data are unbalanced. In other words, traffic flows are typically heavy in few hours of a day. Therefore, our data while traffic was heavy are significantly less than the light traffic data. In order to avoid biased training, we used a regularization equation for our loss as follows:

$$Loss = \frac{\sqrt{\sum_{i=1}^{N}[(X_i - Y_i)^2 + \omega_i \alpha]}}{N} \tag{4.1}$$

where α is the absolute value of $X_i - Y_i$, and ω_i is as follows:

$$\omega_i = \begin{cases} 0, & if \quad Y_i > 0.5 \\ 1, & otherwise \end{cases}$$

Each of the proposed networks are trained for 30 epochs over the trained dataset containing the flow information for 20 Network points over of the course of 2 months.

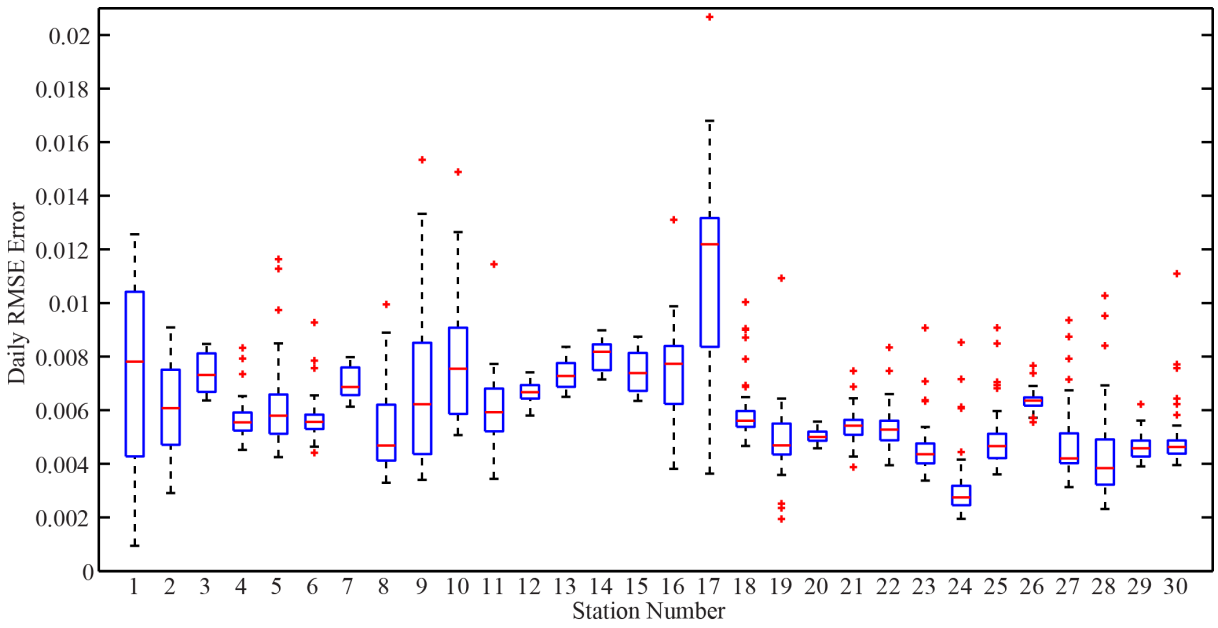

Figure 4.7: Daily RMSE Error for LSTM

The trained networks are then used to predict congestion conditions of the road network.

Fig. 4.6 illustrates the Root Mean Square error (RMSE) error for 30 Network points. The RMSE is computed for each day at a specific Network point and the distribution of RMSE error for each Network point is plotted as a box plot in Fig. 4.6 for the CNN. On the other hand, Fig. 4.7 presents the same RMSE plot for LSTM network. As these plots illustrate, the RMSE for CNN is lower than LSTM and thus results in lower standard deviation.

In order to further evaluate the performance of the proposed methods on the benchmark, we examine the prediction of each network over the course of a day. Fig. 4.8 presents the predicted normalized average speeds for CNN and LSTM networks as well as the ground-truth data. The proposed methods successfully predict the congestion condition with high accuracy in compared with the ground-truth data. However, the error increases slightly during the rush hours and that is due to the un-

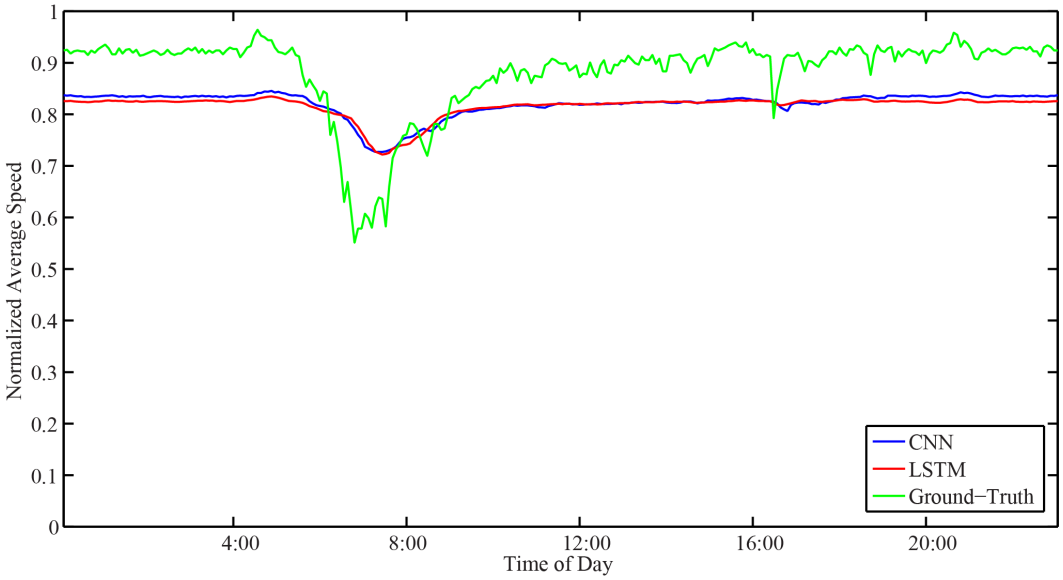

Figure 4.8: Traffic Prediction over the course of a day for a single Network point

balanced nature of the dataset. However, The proposed regularization term effectively decreased the gap during the rush hour.

Hereby, we take a deeper look into the prediction during the rush hours. Fig. 4.9 illustrates the congestion prediction for a single Network point over 30 consecutive days. For this experiment we utilized the information for Network point number 21 at 7:30 am. As shown in this figure, the error increases as the congestion increases however, both networks successfully follow the trends.

On the contrary, Fig. 4.10 illustrates the predicted network flow using LSTM and CNN for light-traffic conditions where we picked 12:00 pm as an example to plot the predicted average speed for Network point number 21. As shown in this figure, the predictions are more accurate in this case.

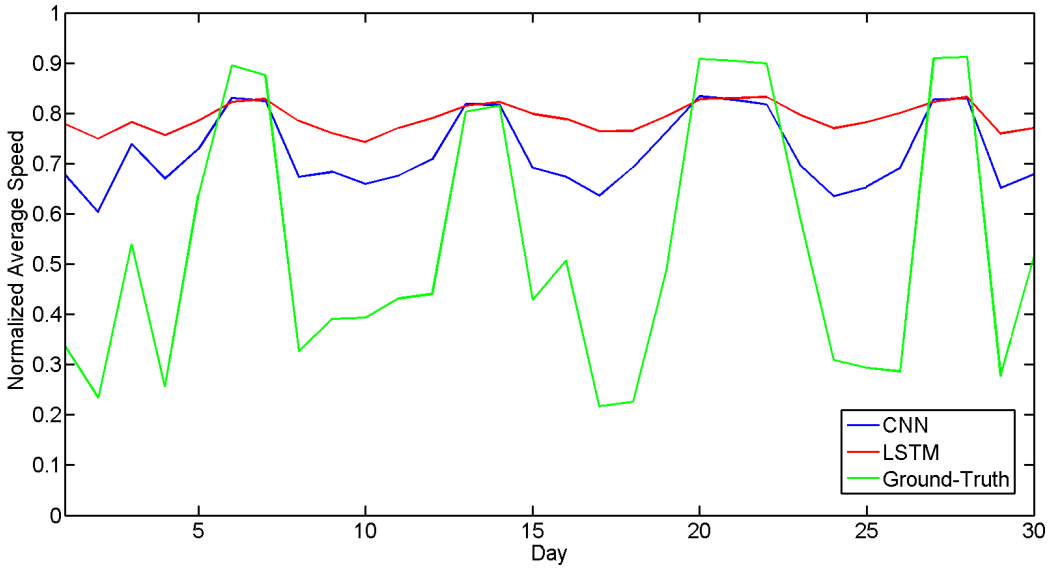

Figure 4.9: Predicted traffic flow during rush hours in 30 consecutive days for a single Network point

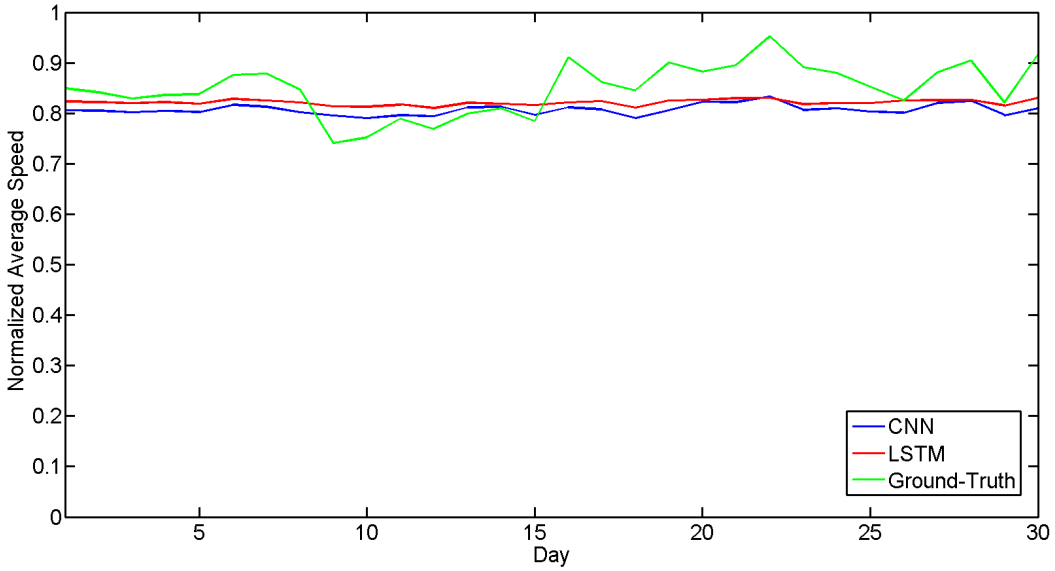

Figure 4.10: Predicted traffic flow during light-traffic hours in 30 consecutive days for a single Network point

4.5 Conclusion

Concepts of traffic bottleneck and congestion propagation are critical components of Intelligent transportation network management systems. There have been lot

of effort to understand how the traffic flows and short-term prediction of congestion occurrence because of rush hours or incidents, such as car crashes or Sport events, can be beneficial to such systems to effectively manage and direct the traffic to the most appropriate detours. Most of traffic flow prediction systems rely on utilizing a central processing component where the prediction is carried out through aggregation of the information gathered from all measuring stations. Nevertheless, such system are typically scalable and unable to provide real-time feedback to the system whereas in a decentralized scheme, each node is responsible to predict its own short-term congestion based on the local current measurements in neighboring nodes.

In this work, we introduced a scalable decentralized traffic flow prediction by utilizing deep learning-based method. Therefore each node accurately predicts its own congestion state in real-time based on the congestion state of the neighboring Network point. Besides, proposed method is significantly suitable in the cases, where we need to predict the traffic flow of newly installed stations, using Deep Network trained by historical data from another traffic network. we introduced a regularized euclidean loss function that favors high congestion samples over low congestion samples to avoid the impact of the unbalanced training dataset. A novel dataset for this purpose was designed based on the traffic data obtained from traffic control stations in northern California. Extensive experiments conducted on the designed benchmark reflected a successful congestion prediction.

CHAPTER 5

Map Matching Using Frechet Distance Algorithm

In this chapter, after having an introduction in Section 5.1, in Section 5.2, we cover a Classic Frechet Distance Problem and a Standard Frechet Distance Algorithm. In the following, in Section 5.3, we come up with some GIS applications for Frechet Distance. Out of these application, we talk about Map Matching and how we can utilize Frechet Distance in this regard. In Section 5.4, we talk about some related work. In Section 5.5, after some formal definitions about frequently used concepts and the problem, we explain our algorithm. In Section 5.7, we talked about the dataset we used to evaluate our work, and we evaluated our approach. Finally, we conclude in Section 5.8.

5.1 Introduction

Concept of polylines similarity, also called curve similarity, is one of the popular *computational geometry* problems, which appears in a variety of different domains, such as machine learning (pattern recognition, speech recognition, signature verification, computer vision, etc.) and GIS applications. One of the most well-known polyline distance metrics is Frechet Distance that was introduced in 1906 [49]. In 1995, [50] came up with an algorithmic solution to compute the Frechet Distance between two polylines.

Frechet Distance can be defined as the minimal length of a leash connecting a dog on one trajectory (polyline) with its owner on a second trajectory, both never

moving backward. Based on this intuitive definition, the GIS applications of Frechet Distance seem more significant than others.

Some of the applications can be:

- **Map Integration and Verification:** Map Integration and Verification is the process of matching two distinct topological datasets that present the same road network.
- **Map Integrity:** Map Integrity is the process of evaluating the accuracy of topological information of a road network by using the GPS data received from GPS devices associated with moving objects who traveled on the aforementioned road network.
- **GPS Tracking or Map Matching:** This process is to match GPS data (location points) generated by a GPS device associated with a moving object to the road network to determine which road the vehicle is on.

5.2 Frechet Distance Problem

Similarity of Polylines has been studied as one of the considerable issues in computational geometry. Among all available metrics, the Hausdorff Distance [51] and Frechet Distance [50] are the most well-known ones. In this research, we utilize the Frechet Distance because the Hausdorff Distance is too static. In other words, the Hausdorff Distance does not support directions. It also does not consider any dynamics of motion along the polylines.

In this section, we briefly cover introduce Frechet Distance algorithm presented in [50]. In order to do this, we define some of the concepts we frequently use in the rest of this chapter. A schematic definition for the Frechet Distance can be as follows:

A man is walking his dog. Obviously, they have different trajectories with different instantaneous speeds. However, both are moving forward (backward move-

ment are not allowed). The Frechet Distance is defined as the minimal leash length that makes this walk possible.

Line Segment: A straight line that is described by two coordinates, starting point (x_s, y_s), and end point(x_e, y_e).

Polyline: A sequence of Line Segments, such that the starting point of the ith line segment is the end point of the (i-1)th line segment. In other words, a polyline is a sequence of points, (x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_n, y_n). Therefore, it is a function P: [0, n] $\rightarrow \mathbb{R}^2$, where n is the length of the polyline and P(i) represents the coordinate of the starting point in the ith line segment. In the rest of this research P_i shows the ith line segment in P.

Frechet Distance: Given two polylines, P and Q, with length of n and m respectively, the Frechet Distance between P and Q is as follows [52]:

$$\delta_F(P,Q) = \inf_{\alpha,\beta} \max_{t\in[0,1]} d(P(\alpha(t)), Q(\beta(t)))$$

Where d is Euclidean distance, and α and β are continuous non-decreasing functions defined as follows:

$$\alpha : [0,1] \rightarrow [0,n], \text{ such that } \alpha(0) = 0 \text{ and } \alpha(1) = n,$$

$$\beta : [0,1] \rightarrow [0,n], \text{ such that } \beta(0) = 0 \text{ and } \beta(1) = m.$$

Free-space Diagram: The free-space diagram between two polylines, P and Q, for a given distance ϵ is the diagram that shows a relationship between all point pairs of P and Q, p and q. If the distance between p and q is equal or less than ϵ, the diagram will be white, otherwise it will be dark. Figure 5.1 demonstrates two polylines, P and Q (a), the distance ϵ (b), and the Free-space between P and Q based on distance ϵ. In Figure 5.1.a, there are two point pairs, represented by blue and red. In Figure 5.1.c, we have the same blue and red points, which represent the same color point pairs as Figure 5.1.a. The blue point is within the dark area which shows that the distance between the blue point pairs is more than ϵ. On the other hand, the red

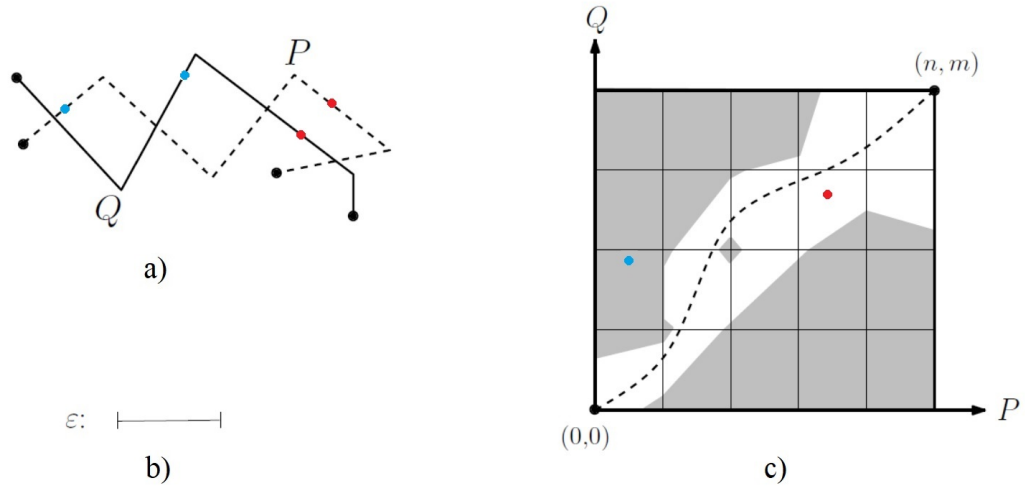

Figure 5.1: a) Polyline P and Q. b) Distance ϵ. c) The Free-space

point is within the white area which shows the distance between the red point pairs is less than (or equal to) ϵ.

The Frechet Distance between P and Q is less than or equal to ϵ if, and only if, we can draw a path from the lower left corner to the upper right corner (in white area), which is monotone both in the horizontal and in the vertical direction (consider Figure 5.1.c).

5.3 GIS application of Frechet Distance

In this section, we introduce 3 GIS applications of Frechet Distance: Map Integration and Verification, Map Integrity, and Map Matching. It is worth mentioning that GIS applications of Frechet Distance are more than what we can cover in this dissertation. We are only mentioning the ones we plan to talk about in this dissertation. The rest may be included in future studies and work.

5.3.1 Map Integration and Verification

Map data is a type of spatial date that is usually collected and maintained for different purposes for use in different applications [53]. For example, they can be gathered and used for creating printed maps. In this case, they do not need to be accurate, as they are usually simplified to make them more readable. On the other hand, the map data used for navigation systems in GPS devices are supposed to be precise and full of informative features, such as speed limits and connectivity flow in routes connected by a junction (described in Chapter 2). This inconsistency in map data needs to be recognized by using Map Integration and Verification. Map Integration and Verification can be accomplished by considering different features of data, such as name or topological information. These differences can be geometric or semantic. For example, consider Figure 5.2. As we see, the map data illustrated by the blue lines show some roads as single lines (one-way)and other roads as two lines (one for each direction of the road), while also using inconsistent naming.

The complex nature of spatial data makes the process of integration and verification more difficult than other types of data, such as texts or numbers. Furthermore, map datasets are very large because they contain information about landmarks, roads, buildings, etc. Therefore, the whole country could easily grow into millions of objects.

5.3.2 Map Integrity

As we discussed in Subsection 5.3.1, map data can be inaccurate and inconsistent. In the cases where we have accurate topological information of road networks (gathered by very precise GPS devices), and we need to evaluate the accuracy of the latter dataset by calculating the *error* between these datasets, we use Frechet Distance. Consider Figure 5.4. It illustrates a road network and a large amount of

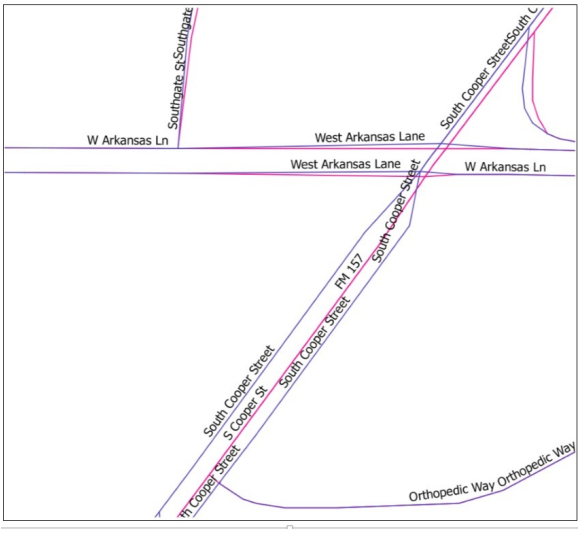

Figure 5.2: Map Integration and Verification

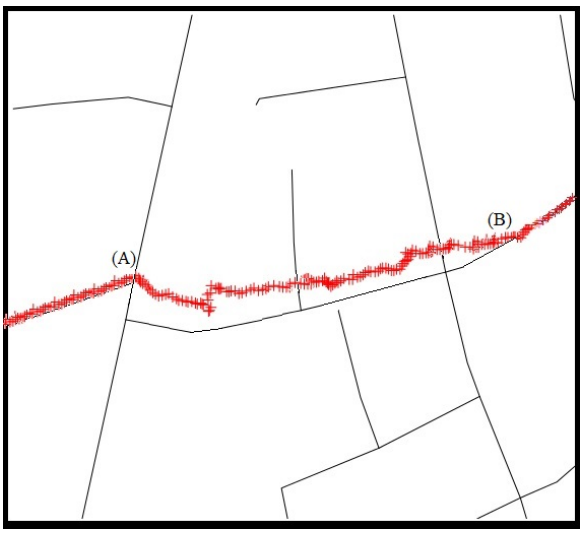

Figure 5.3: Map Integrity

GPS points on the road network. As we can see, the GPS information received from the moving objects from point (A) to point (B)do not match with the road network they are moving on.

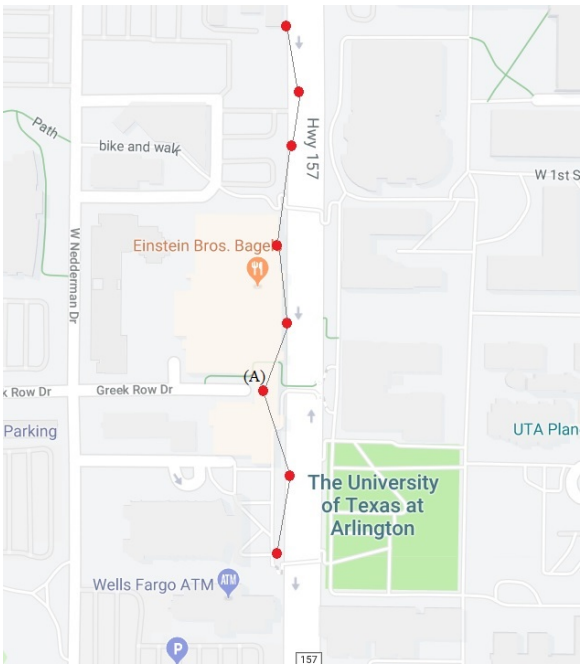

Figure 5.4: Map Matching

5.3.3 Map Matching

The popularity of GPS devices in recent years lead the production of a large amount of raw coordinate data. This data typically has positioning errors. In other words, the latitudes and the longitudes recieved by the GPS devices usually do not completely match any routes along the road network where the moving object associated with the GPS device traveled. GPS satellites broadcast their signal with a certain accuracy, but what a GPS device receives and sends depends on other factors such as atmospheric condition and device quality. For example, GPS enabled smart phones are typically accurate to within 5 meters [54]. However, their accuracy worsens near buildings, bridges, and trees. The general purpose of Map Matching is to match the raw GPS data to road segments, where the moving object associated the GPS device traveled. Map Matching is not just based on the closeness of a point to a road segment, but we need to determine the best match based on the previous and

next location points on the trajectory. For example, Figure 5.4 shows the trajectory of a moving object that is traveling on *Highway 157*. As we see, point *(A)* is close to *Greek Row Drive*, although based on the rest of the trajectory points, it is obvious that point *(A)* is not representing a point on *Greek Row Drive*, but a point on *Highway 157*.

5.4 Related Work

With the advent of smart phones and strong navigation systems, the issue of matching the GPS points to the correct location on the road network has become more significant. In this regard, different Map Matching algorithms have come into picture. These algorithms can be categorized based on various criteria. One of these categorizations classifies these algorithms based on batching the data. *Local/Incremental* algorithms [55, 56, 57] use a small portion of the trajectory for map matching rather than entire trajectory units and as a result, they are fast and perform well. Other algorithms in this categorization are called *global* algorithms [58, 59, 60, 61, 62]. They process all trajectory points in the map matching process. Therefore, they are more accurate in the case of a low sampling rate. In our work, we use the advantages of both approaches. Thus, our approach is fast and it is accurate for low sampling rates.

In another classification, map matching can be grouped into four categorizations. The first group is called geometric which solely uses the geometric information of the road network without considering the connectivity between trajectory units. The second group of algorithms, called topological map matching, considers the connectivity between consecutive trajectory units. The third group of algorithms are based on probability and called probabilistic. The last group uses advanced algorithms for map matching, such as [63] uses Kalman filter, [64] utilizes fuzzy logic, and [65] applys Hidden Markov Model. Our approach can be considered a combina-

tion of geometric and topological because we find the nearest route(s) to a trajectory unit (geometric) and then based on the relationship between trajectory units, we determine which routes are selected for route(s) (topological).

5.5 Preliminaries

In this Section, we give a formal definition to a Map Matching problem in Subsection 5.5.1. Then in section 5.5.2, we formally introduce the problem presented in this work.

5.5.1 Formal Definition

A traffic network comprises a set of roads, as well as set of junctions. Junctions can be intersections of streets, exit-entrance of highways, roundabouts, beginning-end points of a road, U-turns, etc. In the subject of Map Matching, we can consider junctions as the main points of a network because these points are used for moving objects to change their directions. Therefore, it helps us make better decisions on our Map Matching problem. In the following, we define these concepts briefly.

Definition 0 (Moving Object). A moving object is an entity that is represented by a time-dependent position point in space, such as cars, trucks, etc. (spatio-temporal).

Definition 1 (Traffic Network). A traffic network (in our context) is a planar directed graph, comprised of a set of routes (Edges) and junctions (Vertices), $\mathcal{N}(Route, Junction)$. Junctions can be intersections of streets, exit-entrances of highways, or roundabouts.

Definition 2 (Moving Object Trajectory). Through location update, a moving object sends a Location Message, which includes the location information

and time. A sequence of this location information sent by the moving object makes a trajectory. Therefore, a trajectory is $lm_1, lm_2, ..., lm_n$, where $lm_i = (x_i, y_i, t_i)$.

Definition 3 (Path). A path in a Traffic Network is defined by two locations on the network, either on a route or on a junction such as s and d. It can also be a sequence of consecutive junctions on the network between s and d, such that a moving object can travel from s to d on the network without the need to pass any other junctions from the network. Therefore, a path π is $s, j_1, j_2, ..., j_n, d$, where $j_i \in Junction$ (Consider Definition 0).

5.5.2 Problem Definition

Assume we have the trajectory of a moving object on a road network, and we need to know the path that the moving object actually traveled on.

5.6 Map Matching with using Frechet Distance Algorithm

In this section, we introduce a new approach for Map Matching. We come up with a new algorithm that takes the trajectory of a moving object as a polyline and a road network $\mathcal{N}$, and by using Frechet distance algorithm, explained in Section 5.2, returns path π on road network $\mathcal{N}$ where the moving object was traveling on. This algorithm is explained in Algorithm 4

Algorithm 4 Map Matching by Using Frechet Distance

```
1: procedure MMDF(T, 𝒩)
2:     for all p_i(x_i, y_i) in polyline T (1 ⩽ i < n do
3:         l_i = list of all line segment < R_s, R_e >, which are located less than ε to p_i
4:         for all pair < r_s, r_e > of line segments in l_i and l_{i+1} do
5:             Find the shortest path and call it π_temp
6:             if π_temp and T_i are satisfying ε Frechet distance then
7:                 add pi_temp to resultGraph
8:     for all pair < r_s, r_e > of line segments in l_1 and l_n do
9:         if There is a path π from r_s to r_e in reslutGraph then
10:            if π and T is satisfying ε Frechet distance then
11:                put π in set of qualifiedResults
```

In Algorithm 4, we have trajectory (polyline) T and road network $\mathcal{N}$ as the input. In lines 1 and 2, for each point in T, we find list of all the routes within ϵ distance from the point. Then in lines 3 - 7, for each pair of routes in two consecutive lists, we find the shortest path between them and call it π_{temp}. If π_{temp} and the corresponding trajectory unit T_i are within ϵ from each other according to the Frechet Distance Algorithm, we add π_{temp} (all the edges of π_{temp}) into *graphReslut*. At the end of these steps, we would have a new graph containing possible paths from source to destination (starting point of T_1 to end point T_n). In lines 8 - 11, for each path π from the source to the destination, we verify that π and T satisfy ϵ Frechet distance, and we return π as (one of) the result(s).

5.7 Experimental Evaluation

In this section, we talk about the dataset we use for the experimental results in Subsection 5.7.1. We also describe our algorithm and how we generate synthetic GPS data. Then in section 5.7.2, we evaluate our Map Matching algorithm presented in Section 5.6.

5.7.1 Data Source

In order to examine the accuracy of our approach, we implemented our algorithm by using a C++ program. Our test environment was a Windows 10 64-bit operating system, Intel Core i7-6500U CPU @ 2.50GHz, with 16GB RAM. The dataset used to evaluate this work cames from BerlinMod, a benchmark for spatio-temporal database management systems (STDBMS) by the University of Hagen [35]. The moving object data were sampled from the simulated behavior of workers commuting between their places of home and work, and additional trips in their leisure time on the street network of the German capital Berlin which contains 3212 routes and

Table 5.1: BerlinMod datasets characteristics

Category	Days	Vehicles	Trip	Moving points
1	2	141	1,797	346,657
2	6	447	15,045	2,998,674
3	13	894	62,510	12,091,785
4	28	2000	292,940	56,129,943

11449 edges [36]. This dataset consists of 4 different sizes of sub-datasets as well as the benchmark queries for them (see Table. 5.1). Since the dataset does not provide the actual route that the moving object traveled on, we needed to generate the trajectory units for each trip. In order to do this, we found the closest route to the starting and end points of each trajectory (trips of the commuters we mentioned above). We then found the shortest path between the starting and the end points. Once we had the exact routes for the trajectory, we used Algorithm 5. In Algorithm

Algorithm 5 TRAJECTORY MAKER

```
1: procedure TrajectoryMaker(π, periodAvg, periodSD, errorAvg, errorDS, t)
2:     for all R_i in π do
3:         length = Length(R_i)
4:         speed = Speed(R_i, t)
5:         portion = begining of R_i
6:         while portion < end of R_i do
7:             err = makeError(errorAvg, errorDS)
8:             result.Push(position + err)
9:             period = Period(periodAvg, periodSD)
10:            delt = speed * period
11:            position = position + delta
```

5, we get the path as one of the inputs. *periodAvg* and *periodSD* are other inputs that indicate the average period of time and the standard deviation between 2 location messages, respectively. We also get *errorAvg* and *errorDS* that show the average and standard deviation of error for the location where it is sent. t is another parameter that shows the time of the day and is used to retrieve the average speed of a moving object at a specific time along the given route.

A path π contains a sequence of routes. We needed to create location update messages at a certain amount of time (called *period*) for each route. The for loop in lines 2 - 10 shows the process that is done for each route. First, in lines 3 and 4 we calculate the length and the average speed for the route, respectively. Then, we start our travel on the route. Our *position* is set to the beginning of the route. We calculate the error (*err*) and save the position with that error. Then we calculate *period*, considering *periodAvg* and *periodSD*. Then, considering *speed* and calculated *period*, we make the *delta* and move the location to be *delta*.

5.7.2 Evaluation

In this subsection, we are going to evaluate our approach based on accuracy. In this regard, we use *Correct Matching Percentage* (CMP) as defined in [61].

$$CMP = \frac{Correct\ Matched\ Point}{Number\ of\ points\ to\ be\ matched} \tag{5.1}$$

As we already mentioned for our experiment we use BerlinMod Dataset [35] summarized in Table 5.1. In category 4 of this dataset, we have 292294 trajectories. With the help of these trajectories, we can calculate the average and standard deviation of Location Update Messages. Also, we consider 5 meters as the average GPS error and 1 meter for the standard deviation [54]. In this experiment, we can assume the maximum speed for the average at anytime. Algorithm 5 can be used as the general GPS generator because it considers the time for the average speed. However, in this experiment, we assume that moving objects travel on the road network as fast as the maximum speed of the road. Although this assumption can have influence on the sampling rate, we ignore this in the current work, and a more accurate experiment can be done in future work.

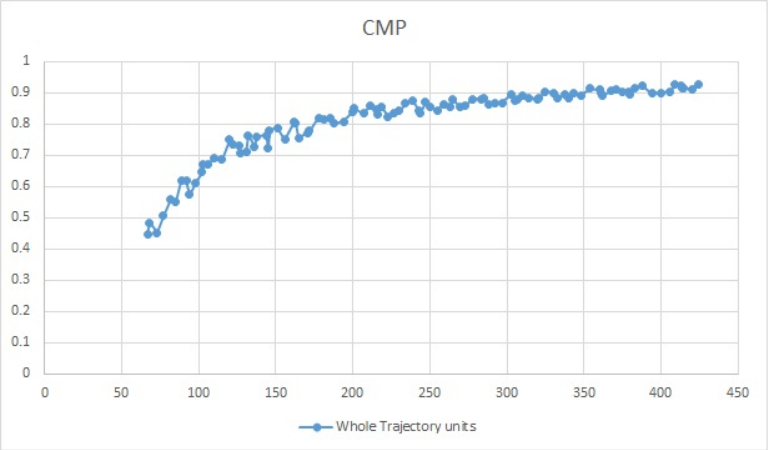

Figure 5.5: CMP for 100 Trajectories

As we mentioned, we use Equation 5.1 to evaluate the accuracy of our work. To do this, we choose 100 pairs of random points from the traffic network (of Berlin City) and find the shortest path between them. Then with the help of Algorithm 5 and aforementioned parameters values, we generate GPS points for those 100 trajectories. Then we sort the trajectories based on the number of their trajectory units from 67 to 424. Figure 5.5 shows the result. In Figure 5.5 the X-axis is the number of trajectory units and the Y-axis is CMP for that trajectory. As we can see, the accuracy increases when the number of sampling points is higher. The reason for this is when we have a higher number of points we have a better ability to determine the actual routes for the (middle) trajectory units. Also, the starting and end trajectory units are usually are assigned to incorrect routes, as we have less insight about their neighbors. In the second experiment, we neglect the first ten and the last ten trajectory units, and again calculate CMP for the same 100 trajectories. Figure 5.6 shows both experiments. The blue dots show CMP before neglecting and the orange ones show after. As we can see, the growth of accuracy for trajectory with less trajectory units is higher. As

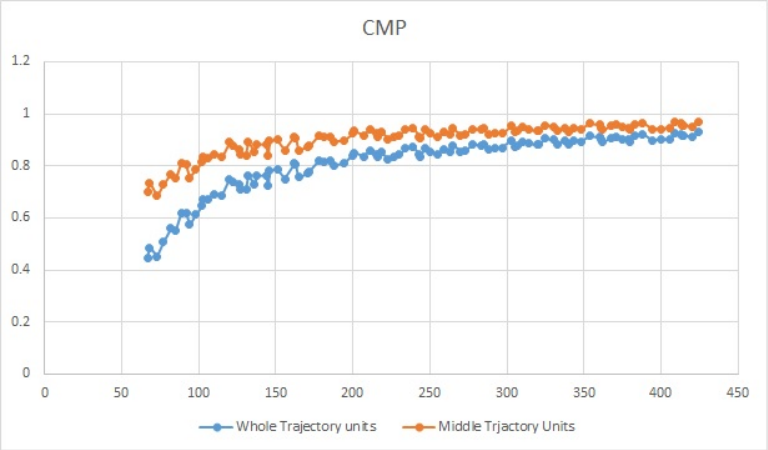

Figure 5.6: CMP for 100 Trajectories after neglecting the first ten beginning and the last ten end trajectory units

mentioned, the beginning and end points of trajectory can be assigned incorrectly because we do not have enough insight about their previous routes.

5.8 Conclusion and Future work

In this work, we presented a new approach to do Map Matching. In other words, we used Frechet Distance algorithm in Map Matching issue. In order to do this, first we described Frechet Distance. Then, we explained Frechet Distance algorithm. In the following, we came up with some GIS applications for Frechet Distance. Out of these application, we talked about Map Matching and how we can utilize Frechet Distance in this regard. After some formal definition about frequently used concepts and the problem, we explained our algorithm. Finally, we talked about the dataset we used to evaluate our work, and we evaluated our approach. As we already mentioned, the Frechet Distance algorithm can be used in variety of different domains, such as machine learning (pattern recognition, speech recognition, signature verification, computer vision, etc.) and GIS applications. In our future work, we want

to use Frechet Distance in other GIS application such as Map Integration and Map Integrity.

CHAPTER 6

CONCLUSIONS

6.1 Summary of Contributions

This dissertation was focused on three aspects of Moving Object Databases: Formalizing of Network-constrained moving objects and their queries and introducing of Temporally Enhanced Network-constrained (TENC) Rtree; Scalable urban traffic congestion prediction by using Deep learning; Map Matching by using Frechet Distance algorithm.

In Chapter 2 and 3, we categorized the various types of Network-constrained moving objects queries based on 4 different aspects of these types of data, namely, *Spatial*, *Temporal*, *Object id*, and the *Path* where the moving objects travel on. Then, we formally defined these categories by using Relational calculus expressions. These categories can be used for 3 purposes:

1. Comparing the performance of indexing methods,
2. Showing the completeness of benchmarks, and
3. Writing the Moving Object Queries easily.

In this regard, we proposed a new indexing method called Temporally Enhanced Network-constrained (TENC) Rtree. This indexing method exponentially shorten the response time to the queries with Temporal, Object id, or/and Path aspect.

In Chapter 4, we tried to predict the traffic flow of road network where we do not have any historical data about them based on the traffic patterns of other road networks where we have their historical data about them.

In Chapter 5, we talked about polylines similarity. Polylines similarity is a very fundamental concept when dealing with Spatial, Spatio-temporal, and Moving object databases. One of Polylines similarity applications is *Map Matching*, where we correspond the location points generated by GPS devices associated with moving objects to the road network to determine which road they are traveling (or traveled) on. In this chapter, we concentrated on a very intuitive measure called Frechet distance with superior quality in theory and practice to solve *Map Matching*. The Frechet distance is defined as the minimal length of a leash connecting to a dog on one trajectory with its owner on a second trajectory, both never moving backwards.

REFERENCES

[1] X. Xu, J. Han, and W. Lu, "Rt-tree: an improved r-tree index structure for spatiotemporal databases," in *Proceedings of the 4th international symposium on spatial data handling*, vol. 2. IGU Commission on GIS, 1990, pp. 1040–1049.

[2] Y. Theodoridis, M. Vazirgiannis, and T. Sellis, "Spatio-temporal indexing for large multimedia applications," in *Multimedia Computing and Systems, 1996., Proceedings of the Third IEEE International Conference on*. IEEE, 1996, pp. 441–448.

[3] M. A. Nascimento, J. R. Silva, and Y. Theodoridis, "Evaluation of access structures for discretely moving points," in *Spatio-Temporal Database Management*. Springer, 1999, pp. 171–189.

[4] M. A. Nascimento and J. R. Silva, "Towards historical r-trees," in *Proceedings of the 1998 ACM symposium on Applied Computing*. ACM, 1998, pp. 235–240.

[5] Y. Tao and D. Papadias, "Efficient historical r-trees," in *Scientific and Statistical Database Management, 2001. SSDBM 2001. Proceedings. Thirteenth International Conference on*. IEEE, 2001, pp. 223–232.

[6] V. T. De Almeida and R. H. Güting, "Indexing the trajectories of moving objects in networks," *GeoInformatica*, vol. 9, no. 1, pp. 33–60, 2005.

[7] E. Frentzos, "Indexing objects moving on fixed networks," in *International Symposium on Spatial and Temporal Databases*. Springer, 2003, pp. 289–305.

[8] X. Meng, Z. Ding, and J. Xu, "Moving objects indexing," in *Moving Objects Management*. Springer, 2014, pp. 51–72.

[9] Y. Theodoridis, "Ten benchmark database queries for location-based services," *The Computer Journal*, vol. 46, no. 6, pp. 713–725, 2003.

[10] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang, "Moving objects databases: Issues and solutions," in *Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on.* IEEE, 1998, pp. 111–122.

[11] A. Guttman, *R-trees: a dynamic index structure for spatial searching.* ACM, 1984, vol. 14, no. 2.

[12] Z. Ding and R. H. Guting, "Managing moving objects on dynamic transportation networks," in *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on.* IEEE, 2004, pp. 287–296.

[13] Z. Ding, B. Yang, R. H. Güting, and Y. Li, "Network-matched trajectory-based moving-object database: Models and applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 1918–1928, 2015.

[14] B. Krogh, N. Pelekis, Y. Theodoridis, and K. Torp, "Path-based queries on trajectory data," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.* ACM, 2014, pp. 341–350.

[15] W. Zheng, D.-H. Lee, and Q. Shi, "Short-term freeway traffic flow prediction: Bayesian combined neural network approach," *Journal of transportation engineering*, vol. 132, no. 2, pp. 114–121, 2006.

[16] X. Zhang and J. A. Rice, "Short-term travel time prediction," *Transportation Research Part C: Emerging Technologies*, vol. 11, no. 3, pp. 187–210, 2003.

[17] L. D. Vanajakshi, B. M. Williams, and L. R. Rilett, "Improved flow-based travel time estimation method from point detector data for freeways," *Journal of Transportation Engineering*, vol. 135, no. 1, pp. 26–36, 2009.

[18] N. Juri, A. Unnikrishnan, and S. Waller, "Integrated traffic simulation-statistical analysis framework for online prediction of freeway travel time," *Transportation Research Record: Journal of the Transportation Research Board*, no. 2039, pp. 24–31, 2007.

[19] M. S. Dougherty, H. R. Kirby, and R. D. Boyle, "The use of neural networks to recognise and predict traffic congestion," *Traffic engineering & control*, vol. 34, no. 6, 1993.

[20] L. Rilett and D. Park, "Direct forecasting of freeway corridor travel times using spectral basis neural networks," *Transportation Research Record: Journal of the Transportation Research Board*, no. 1752, pp. 140–147, 2001.

[21] H. Yin, S. Wong, J. Xu, and C. Wong, "Urban traffic flow prediction using a fuzzy-neural approach," *Transportation Research Part C: Emerging Technologies*, vol. 10, no. 2, pp. 85–98, 2002.

[22] C. Kuchipudi and S. Chien, "Development of a hybrid model for dynamic travel-time prediction," *Transportation Research Record: Journal of the Transportation Research Board*, no. 1855, pp. 22–31, 2003.

[23] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, "Sequential deep learning for human action recognition," in *International Workshop on Human Behavior Understanding*. Springer, 2011, pp. 29–39.

[24] X. Ma, H. Yu, Y. Wang, and Y. Wang, "Large-scale transportation network congestion evolution prediction using deep learning theory," *PloS one*, vol. 10, no. 3, p. e0119044, 2015.

[25] Z. Ding, "Utr-tree: An index structure for the full uncertain trajectories of network-constrained moving objects," in *The Ninth International Conference on Mobile Data Management (mdm 2008)*. IEEE, 2008, pp. 33–40.

[26] R. H. Güting, T. de Almeida, and Z. Ding, "Modeling and querying moving objects in networks," *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases*, vol. 15, no. 2, pp. 165–190, 2006.

[27] S. Abraham and P. S. Lal, "Spatio-temporal similarity of network-constrained moving object trajectories using sequence alignment of travel locations," *Transportation research part C: emerging technologies*, vol. 23, pp. 109–123, 2012.

[28] R. H. Güting and M. Schneider, *Moving objects databases*. Elsevier, 2005.

[29] B. Kuijpers and W. Othman, "Trajectory databases: Data models, uncertainty and complete query languages," *Journal of Computer and System Sciences*, vol. 76, no. 7, pp. 538–560, 2010.

[30] Z. Ding and R. H. Güting, "Uncertainty management for network constrained moving objects," in *International Conference on Database and Expert Systems Applications*. Springer, 2004, pp. 411–421.

[31] Z. Ding and X. Zhou, "Location update strategies for network-constrained moving objects," in *International Conference on Database Systems for Advanced Applications*. Springer, 2008, pp. 644–652.

[32] Y. Tang, X. Ye, and N. Tang, *Temporal Information Processing Technology and Its Application*. Springer, 2011.

[33] I. Sandu Popa, K. Zeitouni, V. Oria, D. Barth, and S. Vial, "Indexing in-network trajectory flows," *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases*, vol. 20, no. 5, pp. 643–669, 2011.

[34] 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*. ACM, 2003, pp. 25–32.

[35] C. Düntgen, T. Behr, and R. H. Güting, "Berlinmod: a benchmark for moving object databases," *The VLDB Journal*, vol. 18, no. 6, pp. 1335–1368, 2009.

[36] ——. (2011) Berlinmod. Http://dna.fernuni-hagen.de/secondo/BerlinMOD/BerlinMOD.html.

[37] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis, "R-trees in spatiotemporal databases," in *R-Trees: Theory and Applications*. Springer, 2006, pp. 99–115.

[38] L.-V. Nguyen-Dinh, W. G. Aref, and M. Mokbel, "Spatio-temporal access methods: Part 2 (2003-2010)," 2010.

[39] T. T. T. Le and B. G. Nickerson, "Efficient search of moving objects on a planar graph," in *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*. ACM, 2008, p. 41.

[40] Y. Fang, J. Cao, Y. Peng, and L. Wang, "Indexing the past, present and future positions of moving objects on fixed networks," in *Computer Science and Software Engineering, 2008 International Conference on*, vol. 4. IEEE, 2008, pp. 524–527.

[41] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[42] C. Goller and A. Kuchler, "Learning task-dependent distributed representations by backpropagation through structure," in *Neural Networks, 1996., IEEE International Conference on*, vol. 1. IEEE, 1996, pp. 347–352.

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[44] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[45] S. of California. (2014) Pems. Http://pems.dot.ca.gov/.

[46] M. D. Laboratory. (2016) Traffic dataset. *https* : *//www.dropbox.com/sh/uo*634*k*3*ybvmu*1*dc/AAAOxRpk* − 2$Q_1$87fZ9$tZmRABa$?$dl = 0$.

[47] S. Dieleman, J. SchlÃijter, C. Raffel, E. Olson, S. K. SÃÿnderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, diogo149, B. McFee, H. Weideman, takacsg84, peterderivaz, Jon, instagibbs, D. K. Rasul, CongLiu, Britefury, and J. Degrave, "Lasagne: First release." Aug. 2015. [Online]. Available: http://dx.doi.org/10.5281/zenodo.27878

[48] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[49] M. M. Fréchet, "Sur quelques points du calcul fonctionnel," *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, vol. 22, no. 1, pp. 1–72, 1906.

[50] H. Alt and M. Godau, "Computing the fréchet distance between two polygonal curves," *International Journal of Computational Geometry & Applications*, vol. 5, no. 01n02, pp. 75–91, 1995.

[51] M. J. Atallah, "A linear time algorithm for the hausdorff distance between convex polygons," *Information processing letters*, vol. 17, no. 4, pp. 207–209, 1983.

[52] A. Maheshwari, J.-R. Sack, K. Shahbaz, and H. Zarrabi-Zadeh, "Fréchet distance with speed limits," *Computational Geometry*, vol. 44, no. 2, pp. 110–120, 2011.

[53] M. Schäfers and U. W. Lipeck, "Simmatching: adaptable road network matching for efficient and scalable spatial data integration," in *Proceedings of the 1st ACM SIGSPATIAL PhD Workshop*. ACM, 2014, p. 5.

[54] F. van Diggelen and P. Enge, "The worlds first gps mooc and worldwide laboratory using smartphones," in *Proceedings of the 28th International Technical*

Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2015), 2015, pp. 361–369.

[55] J. S. Greenfeld, "Matching gps observations to locations on a digital map," in *81th annual meeting of the transportation research board*, vol. 1, no. 3, 2002, pp. 164–173.

[56] A. Civilis, C. S. Jensen, J. Nenortaite, and S. Pakalnis, "Efficient tracking of moving objects with precision guarantees," in *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on.* IEEE, 2004, pp. 164–173.

[57] A. Civilis, C. S. Jensen, and S. Pakalnis, "Techniques for efficient road-network-based tracking of moving objects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 5, pp. 698–712, 2005.

[58] H. Alt, A. Efrat, G. Rote, and C. Wenk, "Matching planar maps," *Journal of Algorithms*, vol. 49, no. 2, pp. 262–283, 2003.

[59] H. Yin and O. Wolfson, "A weight-based map matching method in moving objects databases," in *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on.* IEEE, 2004, pp. 437–438.

[60] 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.* VLDB Endowment, 2005, pp. 853–864.

[61] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G.-Z. Sun, "An interactive-voting based map matching algorithm," in *Mobile Data Management (MDM), 2010 Eleventh International Conference on.* IEEE, 2010, pp. 43–52.

[62] G. Hu, J. Shao, F. Liu, Y. Wang, and H. T. Shen, "If-matching: Towards accurate map-matching with information fusion," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 1, pp. 114–127, 2017.

[63] D. Obradovic, H. Lenz, and M. Schupfner, "Fusion of map and sensor data in a modern car navigation system," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 45, no. 1-2, pp. 111–122, 2006.

[64] M. A. Quddus, R. B. Noland, and W. Y. Ochieng, "A high accuracy fuzzy logic based map matching algorithm for road transport," *Journal of Intelligent Transportation Systems*, vol. 10, no. 3, pp. 103–115, 2006.

[65] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*. ACM, 2009, pp. 336–343.