Spatial query processing for location based services

Liu, Yaqiong

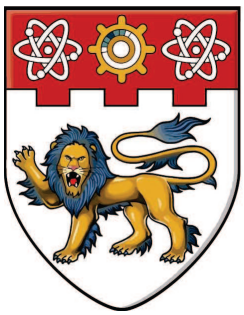
2016

Liu, Y. (2016). Spatial query processing for location based services. Doctoral thesis, Nanyang Technological University, Singapore.

https://hdl.handle.net/10356/66553

https://doi.org/10.32657/10356/66553

SPATIAL QUERY PROCESSING FOR LOCATION BASED SERVICES



NANYANG TECHNOLOGICAL UNIVERSITY

SPATIAL QUERY PROCESSING

FOR LOCATION BASED SERVICES

LIU YAQIONG

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

2016

LIU YAQIONG

2016

SPATIAL QUERY PROCESSING FOR LOCATION BASED SERVICES

LIU YAQIONG

LIU YAQIONG

School of Computer Science and Engineering

A thesis submitted to Nanyang Technological University

in partial fulfillment of the requirement for the degree of

Doctor of Philosophy

2016

Acknowledgements

The period of time that I spent at Nanyang Technological University certainly brought major transformations to my entire life as a result of friendship, team work, persistence and intellectual education. I have gained much more than I originally anticipated during the past five years in NTU.

At the very first, my deepest gratitude goes to my main supervisor Professor Seah Hock Soon who provides me with an amazing opportunity to study and do research in his lab. He expertly guides me through my education, shares the excitement of research with us post-graduate students, and gives students full support to scientific exploration. His expertise in how to do research improves my research quality and prepares me for future challenges. He is very amiable and he is not only a venerable professor, but also like a very respected elder of the family, like a benevolent father. His amiability and earnest instructions sink into my mind forever. I would especially like to express my sincere gratitude for his invaluable constructive feedback on my work and for guiding me through the difficult times that I have encountered during my life and research.

I would also especially like to express my sincere appreciation to Professor Cong Gao for his invaluable supervision, unfailing enthusiasm and respectful professionalism in helping me find suitable research directions and in guiding me through the challenges during the long research period. His rich knowledge and honorable personality impress me very much. I have learnt abundant knowledge related to the research and gained a good deal of enlightenment from him. Without his bountiful help, I would not have achieved much progress so fast.

My sincere thanks are extended to all other professors who have given me much comprehensive knowledge during the past five years. I have benefited so much from their courses and encouragement. I would never forget their edification and inspiration.

I am also thankful to all my dear friends: Xin Cao, Budianto Tandianus, Quan Yuan, Hong Ze Liew, Chi Zhang, Yangyang Li, Lei Wei, Phuong-Quynh Dao, Shanshan Feng, Zongyang Ma, Kaiyu Feng, Yi-Chun Wu, Hui-Earn Tan, Xiaoqun Wu, Ming Zhao, Tianyi Wang, Minqi Zhang, Xiaoning Wang, Yusha Li, Yaqing Hou, Peyyi Chee, Xiansen Meng,

Yilang Jiang, *et al.*, for their good company and encouragement. I wish to take this occasion to thank all of them and hope they can achieve more progress during their study or working lives.

Many thanks to my beloved parents for their unconditional care, encouragement, and support. Whenever I encounter difficulties, they are always there cheering me up.

In addition, I am very grateful to School of Computer Engineering for providing excellent facilities and environment for us Ph.D. students. I feel privileged to have been able to study as a Ph.D. student of this school. I would like to show my sincere thanks to all the members of this school. Their company and friendship have certainly lightened up my life. I would never forget their caring.

Finally, I would like to give my sincere thanks to Nanyang Technological University and Ministry of Education of Singapore for the financial support provided for this research.

Contents

List of Tables

List of Figures

Summary

With the rapid development of wireless communications, location-based services (LBSs) have received extensive researchers' attention. Many LBSs are dependent on the tracking of continuously varying positions of a large set of mobile users, which are also called mobile objects or clients. In this thesis, we consider three spatial queries based on three related LBSs.

First, given a road network, a group of moving objects together with their friendships, and a network distance threshold for each pair of friends, the problem of proximity detection in road networks is to find friend pairs whose network distance is within the threshold. The problem of proximity detection is often encountered in massively multiplayer virtual games and friend-locator applications. Because of the limited battery power and bandwidth, we need a solution which incurs low communication cost. Hence, the primary goal of this problem is to reduce the total communication cost. However, most existing proximity detection solutions focus on the Euclidean space which cannot be used in road network space and the solutions for road networks incur substantial communication costs. Motivated by this, we propose two types of solutions to the proximity detection problem in road networks. In the first type of solution, each mobile client is assigned a mobile region of a fixed size. We design algorithms with a fixed radius for the server and client respectively, with the purpose of reducing unnecessary probing messages and updating messages. Second, we present a self-adjustment strategy to automatically adjust the size of the mobile region for the purpose of minimizing the communication cost. Experiments show that our second type of solution works efficiently and robustly with a much lower communication cost with respect to various parameters. In addition, we propose server-side computational cost optimization techniques to reduce the total computational cost.

Second, points of interest (POI) recommendation with real-world applications is another research issue that has attracted researchers' much attention. Given a set of moving users, i.e., moving objects, and their historical GPS trajectories, the POI recommendation problem is to recommend semantic POIs, based on the GPS trajectories of the users. We first develop a novel algorithm, namely, SEM-DTBJ-Cluster, which stands for

semantics-enhanced density-based clustering, for extracting semantic points of interest from GPS trajectories. We subsequently take three different factors (popularity, temporal influence and geographical influence) into consideration, and describe the impacts of popularity, temporal and geographical information, by deriving three different scoring functions based on three recommendation models. We finally combine the three scoring functions together and obtain a unified framework PTG-Recommend for recommending candidate POIs to a moving user. Our work is the first that considers the influence of popularity, the influence of temporal features, and the influence of geographical features of a POI together. Results of experiments on two GPS trajectory datasets not only prove the effectiveness of our framework, but also show that it performs better than the baseline POI recommendation methods with regard to precision and recall.

Third, route queries are important problems that find applications in many location-based services. Considerable existing studies address routing problems in static road networks. However, the travel costs on edges in road networks always change over time. Such road networks are referred as time-dependent road networks. Most existing studies on time-dependent road networks focus on simply finding a shortest path with the minimum travel time without considering waiting at some nodes, or fuel consumption and toll fee. In fact, waiting at a node is likely to happen and one edge can be traversed at different speeds. Additionally, traveling along a route consumes both fuels and toll fee. In many cases, an optimal route is the minimum-cost route under time and speed constraints. Motivated by this, we study the Cost-Optimal routing problem in Time-dEpendent Road networks with time and speed constraints, denoted as COTER for short, where the travel cost of a route is composed of fuel cost and toll fees. We utilize two fuel consumption models and compute the minimum fuel consumption of an edge under the constraint that the travel time on this edge is exactly the given time, for helping users determine optimal speeds on each edge. We allow the toll fee to be an arbitrary single-valued time-dependent function of the departure time for each edge. We define a time-dependent OC function (Optimal Cost function) for each node n_i, and derive the recurrence relation formula between n_i's incoming neighbors' OC-functions and n_i's OC-function. We pro-

pose a five-step approximate algorithm, namely, ALG-COTER, which answers COTER using optimized single-source shortest-path algorithm, topological sorting, dynamic programming, nonlinear programming and backtrack algorithms. Experimental results show that our algorithm answers COTER queries efficiently and our algorithm is scalable with respect to different parameters that have influences on the running time.

Chapter 1

Introduction

In this chapter, we first present the background of location-based services (LBSs). We subsequently present the architecture and the relation among three LBSs-based spatial queries. Then, we present the motivations, problem statements, objectives and outlines of proposed approaches, of these three spatial queries. Finally we outline the contributions and organization of this thesis.

1.1 Background: Location-Based Services

Location-based Services (LBSs) refer to services that provide location-based information to mobile users. They are dependent on positional information associated with the mobile users. They might also depend on other factors, e.g., users' interests or preferences [28]. LBSs take the geographical positions of the moving objects into account, for creating, compiling, combining, filtering or selecting information to users. Location information transmitted to a user refers to his own position, and that is why most LBSs are regarded as self-referencing services.

LBSs offer customized information by considering mobile users' locations. LBSs are so appealing that they motivate the development of numerous applications, such as location-aware services (e.g., proximity detection), tracking services (e.g., tracking people of importance, or vehicles), finder services (e.g., locating points of interest like museums or auditorium), navigation services (e.g., location-dependent yellow pages, or digital travel assistants), and emergency services (e.g., roadside assistance) [59].

An LBS may inform its users of a traffic jam which might affect many users. A user might be informed of the current whereabouts of his/her friends. Some other LBSs may trace the positions of public transport, hazardous materials, security personnel, police cars, or emergency vehicles. The "catch the monster" game, which is a typical example of more developed LBSs, may allow a set of users to work together, for the purpose of surrounding and catching a geo-positioned monster in the virtual world. Other LBSs may provide mobile users with useful information, e.g., to find an appropriate route in road networks.

1.2 Three LBSs-Based Spatial Queries

1.2.1 The Client-Server Architecture for Three Queries

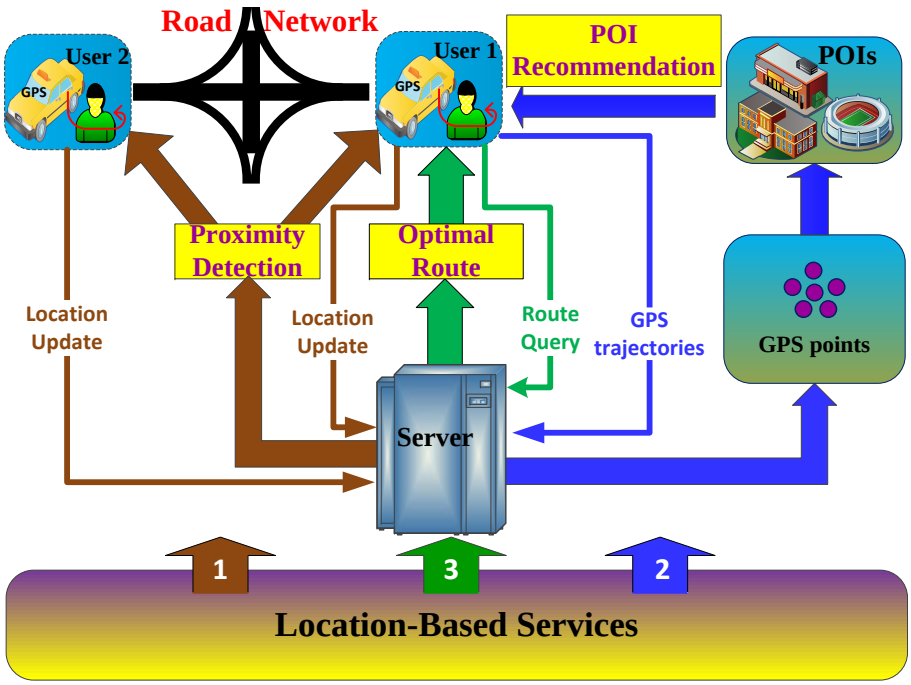


Figure 1.1: Three LBSs-based spatial queries. (The brown flow, blue flow, and green flow represent (1) the proximity detection problem, (2) the POI recommendation problem, and (3) the optimal routing problem, respectively.)

As shown in Fig. 1.1, three LBSs-based spatial queries are depicted. The first spatial query problem is proximity detection in road network. User 1 and User 2 send their location update messages to the central server. The server detects whether they are in

proximity or not. The second spatial query problem is points of interest recommendation. The server clusters GPS points into semantic POIs, and recommends POIs to a user according to a certain recommendation method. The third spatial query problem is optimal route query. The user wants to find an optimal route and the server computes a cost-optimal route for this user and returns the optimal route to him.

1.2.2 The Relations between Three Spatial Queries

The three spatial query problems are all based on location-based services. For each of these three problems, the user sends queries to the server while the server solves the queries and finally replies the user with the query results.

In addition, all these three spatial queries find plenty of applications in both the real world and computer games. For instance, proximity detection is useful for real-world traffic safety; and in some computer games such as the Counter-Strike game, a soldier should be always aware of his friend who is in proximity with him. Take the second spatial query for another example, a traveller may prefer those POIs that are recommended by a POI recommendation system; and it is also possible that an interesting computer game uses some POI recommendation technique to advise a virtual user to visit some POIs. For the third spatial query, optimal route query, no matter in our real-world road networks or the virtual road networks in some computer games, it is very common that a driver queries an optimal route which consumes the least total cost from a source to a destination.

Moreover, all these three spatial queries are based on network graphs. The first and the third spatial queries are based on road network graphs, while the second spatial query is based on a POI network graph if we regard each POI as a node and connect two POIs using an edge.

To evaluate the performance of our approaches to these three queries, we use several road network datasets and GPS trajectory datasets. The method for detecting the proximity relations among mobile users for the first spatial query, can also be used to solve the second query, e.g., judging whether two users have similar geographical/regional preferences if one user always checks in stay points which are in proximity with those

stay points checked in by the second user. As the first spatial query and the third spatial query are both based on road network graphs, we can use similar road network datasets to evaluate the performances of our solutions to these two queries.

1.3 Proximity Detection in Road Networks

In this section, we present the first LBS related spatial query problem, i.e., proximity detection in road networks.

1.3.1 Background

Proximity detection is an important advanced LBS function. The function of proximity detection is an LBS which automatically and continuously detects whether two users inside a set of moving users approach each other towards a pre-defined proximity distance. In Euclidean space, the definition of the proximity detection problem is as below. Given a group of moving objects, a social network G_s describing their friendship, as well as an Euclidean distance threshold $\epsilon_{i,j}$ per friend pair, the server reports whether each pair $\langle u_i, u_j \rangle$ that satisfies two conditions: (i) the objects u_i and u_j are neighbors in G_s, and (ii) *dist*(u_i, u_j) which denotes the Euclidean distance between u_i and u_j is at most $\epsilon_{i,j}$. Representative applications include child tracking, dating services, fleet management and logistics, mobile gaming, and instant messaging, etc. In addition, proximity detection is the foundation of higher-level queries, e.g., cluster detection or convoy query, which checks whether over two users stay close by and is the generalization of the proximity detection query.

1.3.2 Motivation

For a person who is walking on a road, does the distance between him and each of his friends exceed a certain threshold? For a car which is running on a road, how to check whether the car and any other car are too close to avoid collision? The questions above are proximity detection queries in road networks.

The proximity detection query associated to moving objects finds popular applications in either the real or the virtual world. Most applications of GIS (Geographic Information System) like traffic mining tasks, traffic network monitoring, and routing applications, require efficient support of proximity queries in large road networks.

Generally, a moving object equipped with a GPS transmitter in a road network can record its geo-positions and the moving objects may have communications with each other or the server, for example, through mobile phones. There are two communication models in general. Many existing studies adopt the client-server model whereas some others use the distributed architecture. In a centralized client-server architecture, the stationary central server can monitor the traffic. A key point for proximity detection is to figure out at each time stamp whether each pair of friends are close by. This can be computed by the central server. The server can probe the objects regarding their exact positions and the moving objects can update their locations to the central server. This is called communications between the server and each client. In a distributed architecture, the mobile clients communicate with each other directly, and judge whether they are in proximity. No matter which architecture we adopt, the most important optimization goal is to reduce the communication cost, because of limited bandwidth and battery power on clients' mobile devices.

Therefore, the challenge is how to cut down the quantity of messages exchanged, between the server and clients under the client-server architecture. If there are a lot of update or probing messages, the communication cost will be definitely high. At the server side, the server periodically checks whether the distance between each pair of friends is within proximity. It is obvious that the probing cost will be definitely high if the server probes each client which has not reported his locations regularly. Therefore it is imperative to prune unnecessary probing messages. At the client side, a straightforward update strategy is immediate update, which means the clients report their new positions to the central server every time they move to new locations. Another simple update strategy is periodic update, which means the client periodically updates his location. The above mentioned two strategies are straightforward but too naive as lots of update messages will

be incurred. Therefore we must develop an efficient update method for the purpose of reducing the update cost.

Most existing solutions to the problem of proximity detection concentrate on Euclidean space. These solutions include distributed solutions like the strips algorithm [5], and centralized solutions like the self-tuning policies applied in Euclidean space proposed by [134]. There are several existing proximity detection strategies based on several update methods. These update methods include immediate update, periodic update, distance-based update and zone-based update and so on. As proximity detection can find many an application in road networks, it is essential to develop proximity solutions in road networks. There are only a few existing solutions addressing the proximity detection problem in road networks. In addition, before our work, there are no automatic tuning techniques for optimizing the communication cost in road networks.

To speed up k-nearest neighbor (k-NN) and distance-range queries, a network graph embedding technique is proposed in [67]. However, they do not provide any solution for the query of proximity detection. A region-based (zone-based) update strategy to monitor proximity continuously in road networks is proposed in [68]. However, their method is different from ours and they also do not propose self-tuning strategies. Therefore, the demand of developing self-tuning policies to minimize the communication cost for the proximity detection query becomes critical.

1.3.3 Problem Statement

Analogous to the definition of the proximity detection problem in Euclidean space, the definition of the proximity detection query in road networks is as follows: Given a group U of mobile objects, a network graph G_s which describes their friendship, a road network G that all the mobile users move along, and a network distance threshold $\epsilon_{i,j}$ per friend pair, the server needs to report each friend pair $\langle u_i, u_j \rangle$ that satisfies the condition that the network distance $dist(u_i, u_j)$ between this friend pair is at most $\epsilon_{i,j}$.

1.3.4 Objectives and Outlines of Proposed Approaches

For the proximity detection query, our objective is to develop efficient solutions that can be used to answer the proximity detection query in road networks so that the communication cost of the solution is largely reduced.

To achieve this objective, we propose the following approaches.

1. We propose update and probing algorithms with fixed-radius mobile regions for the clients and server in road networks. In a client-server architecture, clients need to report to the server about their locations, speed and other movement parameters. Meanwhile, the server needs to check periodically (e.g., each period equals ΔT) whether these moving objects are within proximity. The server also probes the objects regarding their locations. A simple and direct strategy is that the clients send update messages to the server every ΔT time or the server probes the clients whenever they fail to provide update. This is simple and direct but inefficient in that it will lead to high communication cost. As a result, we propose a mobile region based update approach in which a client will not report his current location to the server as long as he has not moved outside his mobile region, to save the update cost. We also propose three lemmas that prune unnecessary probing messages that are sent by the server by making use of the lower bound and upper bound of the network distance between the mobile regions of two clients, with the purpose of reducing the probing cost.

2. We propose a self-tuning policy to further reduce the total communication cost. We have designed algorithms for the clients and server respectively as mentioned in the first approach. However, with fixed-radius mobile regions, a small radius induces a large update cost while a large radius induces a large probing cost and therefore, there must be such a radius that minimizes the total cost. In order to minimize the communication cost more automatically, we subsequently propose a self-tuning policy which can automatically adjust the radius of each mobile region, by performing expansion and contraction, to reduce the update and probing costs,

respectively.

1.4 Points of Interest Recommendation From GPS Trajectories

In this section, we present the second LBS related spatial query problem, i.e., points of interest recommendation from GPS trajectories.

1.4.1 Background

The pervasiveness of mobile devices and LBSs is leading to an increasing volume of mobility data [89]. Recently the success of *location-aware mobile devices* and *location-aware applications* has fueled more LBSs. During the past few decades, there is a substantial increase in the number of mobile devices together with the utilization of wireless communication, such as GPRS, WI-FI, and Bluetooth. These devices are usually equipped with position sensors with GPS (Global Positioning Systems), which can precisely figure out the locations of the devices. For instance, GPS-enabled devices record their latitude-longitude positions and transfer these trajectory records to a central server. With these devices, individuals are able to acquire their current locations, discover personal important places close to them and plan convenient routes to a destination. The ubiquity of such pervasive technologies results in increasing availability of a huge amount of GPS trajectory data ([89]). For example, many users like recording their open-air activities to form GPS trajectories, for life logging, movement analysis, travel experience sharing, or multimedia content management, etc. Meantime, with forums or Websites appearing on the Internet, people can establish geo-related Web communities. After their GPS logs are uploaded to the communities, people can visualize and manage their GPS trajectories on a Web map. By sharing the GPS logs among each other, people are also able to gain reference information from other people's travel trajectories. For example, a user can find from other users' travel trajectories some appealing places, and thus, design a fulfilled trip for himself.

It is straightforward that people have larger probability of enjoying high-quality travel experience while saving more time for location discovery, if they are given the recommendation suggestions of the interesting places or say, points of interest and travel sequences around them. A point of interest (POI) refers to a concentrated geographic location, e.g., a landmark, a museum, a building, or a restaurant .

1.4.2 Motivation

A majority of existing systems for POI recommendations, such as travel booking Websites [98] focus recommending POIs based on the set of POIs which are given, instead of based on raw GPS trajectories. However, a huge number of GPS trajectories have been accumulating continuously in Web communities, due to the pervasiveness of the GPS-enabled devices. Therefore POI recommendation from GPS trajectoies has significant meanings for the research field of POI recommendation.

It is notable that many LBSs still use raw GPS data directly, like time stamps and coordinates, without enough semantic information. Therefore, such LBSs cannot provide much support when giving users suggestions on geographical locations. Thus, we need to design a framework to give users efficient recommendation on POIs, from GPS trajectories.

Many researches on data mining have studied the problem of recommending a location which a user might make as his next destination. There are works on location identification and recommendation from numerous users' GPS trajectories during a long period. Zheng *et al.* [143] take the lead in the area of mining and ranking locations from GPS data. In [142], they extract interesting travel sequences and locations, which we can recommend to many users.

In [129], Ye *et al.* address the problem which is most closely relevant to our work. They adapt the CF (collaborative filtering) model for POI recommendations, to gain an improvement of the recommendation accuracy. They model the spacial influence by using a CF based Bayesian method. However, their assumptions and techniques are significantly different from our work. In addition, our framework is the first one that tackles POI

recommendation queries by combining popularity, temporal, and geographical influences.

1.4.3 Problem Statement

The POI recommendation query is defined as below. Given a group U of moving users, and their GPS trajectories *Traj*, our objective is to recommend semantic POIs l to a user, based on users' trajectory patterns, by utilizing the influence of popularities of POIs, the influence of the temporal features of POIs, and the influence of the geographical features of POIs.

1.4.4 Objectives and Proposed Approaches

For POI recommendation query, we take not only the temporal influence into consideration, but also the semantics along with the geographical influence into account. Our objective is to extract semantic POIs from GPS trajectories and develop a unified solution that combines the popularity, temporal and geographic influences together to answer the POI recommendation query for each mobile user.

To achieve this objective, we propose the following approaches.

1. We propose a semantics-enhanced clustering algorithm, SEM-DTBJ-Cluster, aiming at extracting semantic POIs from GPS trajectories.

2. We analyze the popularity influence, temporal influence as well as geographical influence from historical GPS trajectories, and derive three scoring functions based on these three kinds of influences. We then propose a novel unified POI recommendation framework, PTG-Recommend, which combines the three scoring functions into one combinatorial scoring function, by which, the unified recommendation score for each POI can be computed.

3. We carry out experiments on two GPS trajectory datasets, aiming at evaluating the performance of our recommendation method PTG-Recommend, with regard to precision and recall.

1.5 Cost-Optimal Route Queries in Time-Dependent Road Networks

This section presents the third LBS related spatial query problem, i.e., cost-optimal route query in time-dependent road networks.

1.5.1 Background

The past decades have witnessed a great many studies in the area of routing in static road networks ([30, 32, 13]). In navigation applications, a foremost query is to find such possible routes along which a user travels from the current point to the target point with the smallest expected cost. A widely adopted metric for evaluating cost is the shortest network distance between the *source* and *target*. Although the classic Dijkstra's algorithm performs well on small graphs, it does not scale well to large graphs. Therefore, more efficient techniques such as [85, 22, 6, 112, 35] have been proposed. These speedup techniques can calculate shortest network distances much faster than Dijkstra's algorithm, even by several orders of magnitude.

Because of weather conditions or traffic jams, road networks are dynamic or time-dependent rather than static. Hence, developing efficient route planning algorithms for time-dependent road networks has become researchers' focus. The smallest cost of a route is greatly dependent upon the departure time. For example, people might change their planned routes to avoid highways during rush hours.

1.5.2 Motivation

Road networks are time-dependent which means edge weights might change over time [34]. It turns out that switching to a time-dependent scenario from a static one is much more challenging. The size of input increases significantly since edge weights on time-dependent road networks often change in different time periods of one day. Further, edge weights are not limited to the length of the edge but extended to toll fee, energy cost, and travel time of an edge. Taking energy consumption into account is a means of saving money and

reducing carbon dioxide emissions. In the study of [118], a model based on VSP (vehicle specific power) is developed to calculate vehicle fuel consumption. [109] utilizes the real-world vehicle operation and emission data to establish a set of fuel consumption models, based on the influences of several driving parameters on emissions and fuel consumption. For the sake of saving energy resources and environmental protection ([112]), users might want to find a route with minimum consumption of fuels. Therefore, it is crucial to develop energy-efficient routing algorithms in time-aware road networks based on these energy consumption models. Meanwhile, toll fee of a road is also time-dependent. For example, according to [74], there are "London congestion charge" and "Road pricing policy" to reduce traffic congestion and control traffic pollution in UK. The vehicles are charged if they pass through the major roads in rush hours. Similar policies are also applied in Singapore [128]. This results in the variation of the toll fee of a road in different hours of a day, which shows the toll fee is time-dependent. Moreover, there are research works that study the pricing mechanism for the time-dependent toll fee ([78]). Another commonly considered metric is the travel time since people certainly prefer a route which takes acceptable time to travel. It is worth noting that vehicles usually have different speeds when running along a road during different time periods of a day. The maximum speed allowed on one road segment is time-dependent. For example, the speed is low in rush hours.

Consider a user who would like to have an appointment with his friends. He might pose such a query:" Find a route r from my home to the destination, so that if I depart at or after 8:00 a.m., I can arrive at the rendezvous before or at 9:00 a.m. Meanwhile, the total cost which includes fuel cost and toll fees can be minimized." In this example, there are three kinds of weights for every road, travel time, fuel cost, and toll fee, which are all time-dependent. This example query has one hard constraint: a budget (travel time) constraint that the route should satisfy. This query attempts to discover the optimal route under this constraint, such that the route has an optimal objective score (e.g., the fuel cost plus toll fee on the route). In the example query, it is likely that the minimum-cost route takes more than 1 hour's travel time. Furthermore, cost-optimal routes are dependent on the departure time. Users might change their planned routes to avoid highways during

rush hours. It is also likely that there may be several candidate routes satisfying the time constraint from the source to the destination. Hence, users want to find an optimal route which has the minimum cost among all candidate routes satisfying the time constraint. In other words, a route searching system should be able to balance the tradeoff between the time budget constraint and the objective score.

1.5.3 Problem Statement

We define the aforementioned type of queries as a Cost-Optimal Time-dEpendent Routing problem, which is denoted as COTER. Given a time-dependent road network G_T, a source n_s, a destination n_e, an earliest departure time stamp t_d from n_s, and a latest arrival time stamp t_a at n_e (which also indicates there is a travel time constraint that the total travel time plus waiting time should be less than $\Delta = t_a - t_d$), find an optimal (most economical) route R from n_s to n_e, satisfying the following three conditions: (i) waiting at a node is allowed and if departing from n_s after time t_d, one can arrive at n_e before time t_a along route R; (ii) the speeds on each edge $e \in R$ satisfy the time-dependent maximum speed constraints; and (iii) route R has the minimum cost (fuel cost plus toll fees) among all the routes satisfying both condition (i) and (ii). To our best knowledge, none of the existing solutions to route queries or path planning are applicable for COTER. Therefore, our problem and solutions are novel.

1.5.4 Objectives and Proposed Approaches

The objective of COTER is to find the optimal route R which has the minimum cost among all the routes from source n_s to destination n_e under the travel time constraint in the given time-dependent road network.

To achieve our objectives, we propose approaches as follows.

- We propose a novel problem, namely, COTER, i.e., Cost-Optimal Time-dEpendent Routing under time and speed constraints, which allows waiting at some nodes.

- We compute the minimum fuel cost of an edge when the travel time of this edge is

given, via nonlinear programming. We allow the toll fee of each road segment to be an arbitrary single-valued function of the departure time. We measure the cost of a route by the sum of fuel cost and toll fees.

- We define an OC (Optimal Cost) function $opt_i(t)$ for each candidate node n_i. We derive the recurrence relation formula between the OC-functions of the incoming neighbors of n_i and the OC-function of n_i.

- We propose a five-step approximate algorithm, namely, ALG-COTER, to solve COTER, by using Fibonacci-heap optimized Dijkstra's algorithm, topological sorting, dynamic programming, minimum heap optimization, nonlinear optimization, and backtracking algorithms. We also analyze the time complexity of ALG-COTER.

- We carry out experiments on three real-world road network datasets, to evaluate the efficiency, sensitivity, and scalability of our ALG-COTER algorithm, by exploring the influences of different parameters on running time.

1.6 Summary of Contributions

In this thesis, three LBSs related spatial query problems are addressed. We summarize the contributions of this thesis in the following outline.

First, we introduce the problem of proximity detection in road networks. We propose two types of solutions based on the client-server architecture to answer the query of proximity detection in road networks. In the first kind of solution, the mobile region for each moving client has a fixed radius. Unless a client moves beyond its mobile region, he has no need to update his location to the server. Additionally, we propose three lemmas for the purpose of reducing the probing cost at the server side. We design the second kind of solution RRMD along with RMD_{RN} and CMD_{RN} methods, by making use of a self-adjustment policy which tunes the radius of the mobile region automatically, so that the total communication cost can be minimized. Experimental results demonstrate that our self-adjustment method can reduce communication cost to a large extent, and is robust

and scalable to various parameters. Moreover, we also use some optimization methods to diminish the server-side computational cost.

Second, we propose a new POI recommendation query, aiming at recommending popularity-temporal-geographical specific POIs for users from GPS data. We propose a novel semantics-enhanced density-based clustering algorithm, namely, SEM-DTBJ-Cluster, to cluster stay points and extract semantic POIs from stay points. We analyze the popularity influence, temporal influence, and geographical influence from users' historical GPS trajectories, and combine the three scoring functions based on these three kinds of influences, to obtain a unified framework, namely, PTG-Recommend (Popularity-Temporal-Geographical-Recommend). Experimental results show that our PTG-Recommend framework can recommend POIs to users with higher accuracy and recall than baseline recommendation methods.

Third, we study the cost-optimal time-dependent route query in road networks. In our setting, we allow waiting at some nodes. By taking time-dependent maximum speed and travel time constraints into account, we utilize two polynomial fuel consumption models and allow an arbitrary toll fee function of the departure time for each edge in a time-dependent road network. We define the OC (Optimal-Cost) function for each node and derive the recurrence relation between the OC function of a node and the OC functions of its incoming neighbors. We subsequently propose an approximate algorithm, namely, ALG-COTER, that can efficiently address this time-aware routing problem. The ALG-COTER finds the optimal route by utilizing the optimized Dijkstra's algorithm, the topological sorting algorithm, as well as dynamic programming, nonlinear programming, and backtracking techniques. We also analyze the time complexity of our ALG-COTER algorithm. Experiments demonstrate that our proposed algorithms are efficient and scalable to different parameters which have influences on the running time.

1.7 Thesis Organization

The structure of this thesis is arranged as follows. The context of related work on LBSs and existing proximity detection techniques as well as existing studies on POI recom-

mendation, and existing works on route queries are reviewed in Chapter 2. Two types of proximity detection solutions in road networks are proposed in Chapter 3. A novel recommendation method, namely, PTG-Recommend is presented in Chapter 4, which takes into consideration three different factors that influence the recommendation score of a POI. Chapter 5 deals with cost-optimal route queries with time and speed constraints in time-dependent road networks. We take time-dependent maximum speed constraint and the travel time constraint, the fuel consumption cost, as well as the toll fee cost into consideration and propose an ALG-COTER algorithm to answer the time-dependent route queries in road networks. Finally, conclusions and ideas for prospective future work are summarized in Chapter 6.

Chapter 2

Literature Review

In this chapter, we first present existing works on Location-based Services and continuous spatial query processing. We subsequently review the related literature on three different LBSs based spatial query problems which are proximity detection problem in road networks, point of interest recommendation problem, and cost-optimal routing problem in time-dependent road networks, respectively.

2.1 Location-Based Services

This section presents related work on LBSs and the LBS position management framework.

2.1.1 Related Work on LBSs

The advancement of mobile devices and the improvement of continuous positioning systems have fueled location-based services [103, 71]. In many cases, LBSs are regarded as reactive services [117], which means location information is transmitted to the user only on demands.

Several recent studies identify LBSs as critical research topics. An architecture to support flexible LBSs while maintaining users' location privacy is proposed in [113]. In this architecture, users' location information is allowed to be shared at different levels of granularity and with different levels of user control. A cost-effective recommender system is developed in [97] for taxi drivers, with the design goal of maximizing their profits

when following the recommended routes for finding passengers. The problem of learning spatial density models and focusing specifically on individual-level data is addressed in [79]. A citywide and real-time model for estimating the travel time of any path (represented as a sequence of consecutive road segments) in real time in a city is proposed in [120]. It is based on GPS trajectories of vehicles received in current time slots and over a period of history as well as map data sources. A large human mobility database (GPS records of 1.6 million users over one year) and several different datasets to capture and analyze human emergency behavior and their mobility after the Great East Japan Earthquake and Fukushima nuclear accident are built up in [108]. They also develop a model of human behavior that takes into account these factors, for accurately predicting human emergency behavior and their mobility following a large-scale disaster. Based on users' daily communication patterns, the work [40] automatically infers users' demographics by making use of the power of big data, on a real-world large mobile network which contains over 1 billion communication records (CALL and SMS) and over 7 million users.

2.1.2 LBS Position Management

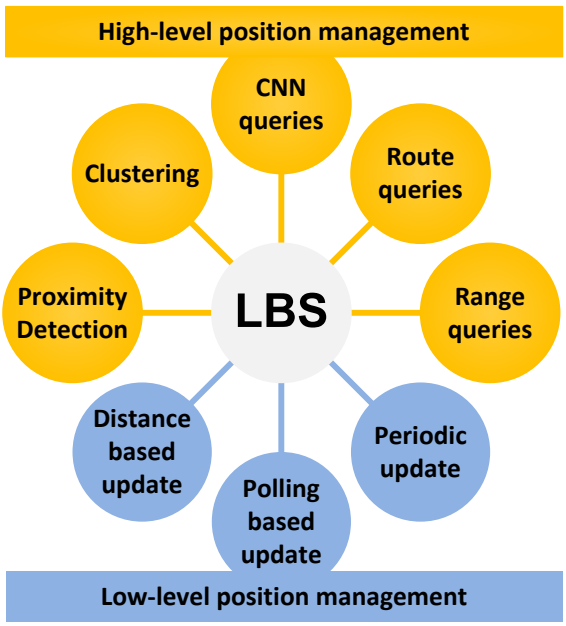


Figure 2.1: LBS position management framework.

The LBS position management framework in [72] is constructed for processing and

collecting positions of multiple clients efficiently. The framework supplies a collection of basic blocks. A wide range of LBS applications applying these basic blocks so that demands of accuracy and up-to-dateness on positions can be satisfied. As depicted in Fig. 2.1, the framework is adapted, in order to be used in conjunction with terminal-based positioning methods and a collection server.

The framework is organized between two layers which represent terminal-based positioning methods (inside the light blue circles) and the LBS applications (inside the orange circles), respectively. The framework is partitioned into high-level and low-level functions. The high-level functions are widely studied in recent years. For example, in [26], the authors propose a UNICONS algorithm for NN queries and CNN queries performed on a network. Proximity detection is another widely studied and applied high-level function, see [5, 64] for example. Some other researchers propose proximity detection methods for medical applications [100]. The low-level functions layer is based on position update methods and supplies distinct methods for position exchanges between the server and a client. The position update methods include polling, periodic update, distance-based update, zone (region) based update, dead reckoning, trajectory updates and query-based updates [75, 72, 117], and so on.

It is desirable to minimize the number of messages induced by the methods while satisfying the requirements of the LBS applications or high-level functions. As a result, different position update methods are adapted and applied on demands.

2.2 Continuous Spatial Query Processing

The past few decades have witnessed a broad study of spatial database, which leads to the development of a huge number of multi-dimensional indices and query processing algorithms [101]. Because of the simplicity and efficiency of R-trees [52, 105, 14], they have become the most widely used indices to process Euclidean queries. For range queries, Q-$index$ [96] utilizes an R-tree to index the ranges. [47] and [19] relieve the burden of the server by utilizing the computing power of the objects. For approximate k-NN queries, [66] proposes a general scheme to reduce information while still answering the ekNN

problem with some error bound, and proposes a technique called DISC (aDaptive Indexing on Streams by space-filling Curves) to facilitate efficient maintenance of K (a user specified constant) points in each cell. For exact k-NN monitoring, at least three methods in the Euclidean space exist: CPM [91], SEA-CNN [125] and YPK-CNN [135]. For proximity queries, [5] proposes a distributed solution called "strips algorithm" in Euclidean space and [134] presents a centralized FMD model together with two self-tuning policies in Euclidean space.

Instant monitoring and predictive evaluation are two categories of server-side continuous spatial query processing methods [134]. On the basis of the unpredictable future locations of queries and objects, instant monitoring is utilized by [91] and [88] to maintain the query results up to date. In a client-server architecture, the server refreshes the results periodically (every ΔT time units), based on update messages sent from the mobile clients. In the instant monitoring category, two representative solutions are CPM [91] and SINA [88] which can efficiently maintain the results of k-NN queries and range queries, respectively, by utilizing a spatial partitioning grid. Predictive evaluation [61, 140] predicts the query results from present to future by modeling clients' prospective locations through linear functions. The server recalculates the future query results corresponding to an object according to the update message sent by the object when the motion function of the object changes. In [61], the authors check temporal events that result in prospective results updates, and develop approaches for maintaining the results of k-NN queries and spatial join queries. All the works above focus on reducing computational cost rather than the expense caused by communication between moving objects and the server. In addition, algorithms of [61, 140] ignore friend pairs and disallow personalized threshold $\epsilon_{i,j}$.

For the purpose of minimizing the communication cost, the idea of *safe region* has been extensively explored. The safe region $SR(p)$ of a client p for static (range or k-NN) queries on moving clients [17, 57, 90], is a zone within which it is guaranteed that the query results are the same. For a query point q, the safe region $SR(q)$ for moving queries on static clients [94, 139], is a zone within which it is guaranteed that the query results

remain the same. [5] proposes a distributed solution where mobile users communicate with one another and a safe region is set up for each friend pair.

In many real-world settings, the mobile clients or queries are usually restricted by a transportation network [92]. However, existing works seldom process queries in road networks. In [95], the authors propose a model which combines Euclidean and network information, and develops a framework that can be performed to the most popular queries successfully. In [26], the authors present an approach called UNICONS to answer the CNN query in road networks. In [92], the authors propose two algorithms that can deal with freeform objects and query moving patterns in a road network. In [24], the SQUARE algorithm is proposed, to construct the network in a way similar to that used in decoupling models, and exploit the coupling idea to maintain the k-NN information relative to areas with frequent queries in the past history.

2.3 Proximity Detection Query

Proximity detection among mobile objects is a well-known problem in the LBS community in recent years. Related work on proximity detection is part of research in the field of LBSs on moving clients (see [2] and [65] for example). There are plenty of research works proposing algorithms for proximity queries. For example, in [93], the authors develop methods for proximity and some other queries in spatial databases.

As one of our spatial query problems is proximity detection in road networks, we present different communication models and existing solutions for proximity detection in road networks in the remainder of this section.

2.3.1 Communication Models in Road Networks

We differentiate between two commonly used communication architectures. One architecture is centralized client-server framework, and the other one is *peer-to-peer* ($P2P$) framework.

In the client-server architecture, clients and the server communicate with each other through messages but no communication is allowed between different clients. At the

client side, each user measures his location periodically (every ΔT time units). Several motion parameters describing locations and predicted prospective movements are maintained at the client side. Clients are allowed to transfer their locations and other motion parameters to a central server. At the server side, the server stores a bunch of users U, motion parameters, a friendship network G, together with a distance threshold $\epsilon_{i,j}$ for each friend pair $\langle u_i, u_j \rangle$. The server has the task of keeping track of locations and friend lists for each client, and is supposed to compute and send probing messages or notification messages to each pair of friends.

In the *peer-to-peer* model [99], no server is involved. Instead, clients are supposed to keep every friend of theirs updated about their locations. The clients will measure their locations in each ΔT time epoch. Each user can update his up-to-date location to each of his friends, or send a probing message to his friends. When the proximity condition is satisfied, the two clients are notified by a proximity alert message.

Most of existing proximity detection methods either adopt the client-server model or the $P2P$ model. For example, [5] adopts distributed $P2P$ architecture; whereas works such as [117, 116, 126, 140, 61, 134] adopt the client-server architecture.

Studies like [110, 111] have proposed several other models such as mobile agents [111, 60], client/agent/server (c/a/s) [8], client/agent/agent/server (c/a/a/s) [102, 123, 9, 56], mixed client and server [62].

2.3.2 Existing Solutions For Proximity Detection Query

Existing proximity detection methods assume that mobile objects are fitted with a cellular mobile devices with GPS enabled either in a client-server [134] or $P2P$ [5] architecture. In a client-server architecture, the client transfers location-based information to a central server; while in a $P2P$ architecture, the client transfers its location information to another client of interest.

As a result of the limited bandwidth and battery power of the clients' mobile devices, reducing the communication cost is one of the most important optimization goals [68]. A straightforward update solution is that mobile clients send update messages periodically,

for example, every second. This is simple but too inefficient. In most of existing works, mobile objects are usually assigned with a region and thus a region (zone) based update strategy is applied. As long as a user locates within the region, no update messages will be triggered. At the same time, in a client-server architecture, the server needs to check whether each pair of users is in proximity or not. Therefore a huge number of probing messages will be caused. To address this problem, the filter-refinement step is applied at the server side.

In [134], the authors adopt the client-server architecture and solve the proximity detection problem in Euclidean space. In their setting, the server detects the friend pairs whose Euclidean distance is within the proximity threshold.

However, in road networks, the problem of proximity detection should be redefined. Only a few existing solutions resolve the proximity detection problem in road networks. In addition, before our work, there are no automatic tuning techniques for optimizing the communication cost in road networks. In [67], a network graph embedding technique for speeding up distance-range and k-NN queries is proposed. However, the key issue in the paper is distance-range and k-NN queries instead of the proximity detection. Therefore they have not given any solutions for proximity detection. A more related work [68] proposes a region-based (zone-based) update strategy for continuous proximity monitoring in road networks. The key technique of this strategy is to maintain a separation region as well as a proximity region for each client and when the client has not moved across the boundary of his region, it is unnecessary to update the proximity/separation results. However, it should be noticed that (i) this method is different from ours (see Chapter 3 for details) although this paper also uses region based update strategy, (ii) our algorithm can deal with road networks with larger number of users, and (iii) they have not proposed self-tuning strategies.

2.4 Points of Interest Recommendation Query

In this section, we first present existing clustering methods and traditional similarity measures which can be adopted when measuring the similarity of two POIs. Subsequently,

we introduce related literature on location identification and recommendation. Finally we review related works on temporal POI recommendation and geographical POI recommendation.

2.4.1 Clustering Algorithms

For gathering POIs from GPS points, we usually perform a clustering algorithm. We mainly introduce three types of clustering approaches: partitioning clustering, time-based clustering, and density-based clustering [144].

2.4.1.1 Partitioning clustering

A typical partitioning clustering algorithm is the famous K-Means. Ashbrook and Starner use K-Means to learn a client's important locations from historical location data [7].

K-Means partitions the total points into K sets, so as to minimize the quadratic sum of the distances from each point to its cluster centroid. K-means is simple and is rather efficient. However, K-Means is sensitive to outliers as the final clustering results contain all the points, even outliers or noise.

2.4.1.2 Time-based clustering

In [54], a time-based clustering method was developed to "extract places" by making use of the continuous WI-FI positioning. Each time the distance between a new location and the previous location is larger than a distance threshold *d*, and meanwhile the new location spans a time threshold *t*, the method discovers a new place of interest.

This algorithm works on mobile devices in a new incremental way. However, this kind of algorithms require continuously collecting location data with very subtle intervals, and therefore much memory is consumed.

2.4.1.3 Density-based clustering

A density-based clustering algorithm named DJ-Cluster is designed by Zhou *et al.* to discover personally meaningful places [43]. DBSCAN [44, 87, 138] is another representative

density-based algorithm. Generally speaking, this kind of algorithms have strong abilities to deal with geographical characteristics, and therefore they are excellent candidates for discovering places.

Density-based clustering can discover clusters of freeform shapes. It especially favors clusters of symmetric shapes like circles or spheres. In addition, it is less probable for noise and unusual points to take part in the final clustering results.

2.4.2 Similarity Measures

Given a set of POIs, we evaluate the similarity of different POIs when making POI recommendations. For example, if a user visits a POI L_i frequently, then we can recommend another POI L_j to him when the similarity between L_i and L_j is high.

This section presents three different methods for measuring similarity, which are similarity-by-count, Pearson similarity, and cosine similarity, respectively.

Similarity by Count. The similarity-by-count method is an intuitive method in that the similarity of two clients is measured by counting the shared regions of the two clients. On Layer i of the shared framework, we generate $N = |C_i|$ ($C_{ij} \in C_i, 1 \leq j \leq N$) clusters. Suppose in cluster C_i, u_1 and u_2 have m_j and m'_j stay points, respectively, then two vectors are expressed as follows.

$$\begin{aligned} U_1 &= \{m_1, m_2, ..., m_j\} \\ U_2 &= \{m_{1'}, m_{2'}, ..., m_{j'}\} \end{aligned} \tag{2.1}$$

The similarity by count is defined as Eq. 2.2.

$$sim_{count} = \sum_{j=1}^{N} \min\left(m_j, m_{j'}\right) \tag{2.2}$$

Pearson Similarity. The metric Pearson similarity [77] varies from -1.0 to +1.0. It measures how highly two variables are correlated. A score of 1 means that these two variables are perfectly correlated while a score of -1 indicates that these two variables are not correlated at all. The Pearson correlation score is a measurement of how well two variables fit a line. In essence, this score computes the ratio between the standard deviation and

the covariance of two objects. This score can be derived in the mathematical form as the following equation:

$$sim_{Pearson} = \frac{\sum xy - \frac{\sum x \sum y}{N}}{\sqrt{(\sum x^2 - \frac{(\sum x)^2}{N})(\sum y^2 - \frac{(\sum y)^2}{N})}} \tag{2.3}$$

Here, N represents the number of properties. (x, y) denotes the data objects.

Cosine Similarity. Cosine similarity [50] is a widely applied metric to measure the similarity of two vectors. Given vectors A and B, we can derive the cosine similarity $cos(\theta)$ between them from the dot product.

$$sim_{cos} = \cos\theta = \frac{A \cdot B}{|A| * |B|} = \frac{A \cdot B}{\sqrt{(A)^2}\sqrt{(B)^2}} \tag{2.4}$$

Cosine similarity ranges from -1 to 1, where -1 means exactly opposite, 0 indicates orthogonality, 1 means exactly the same, and in-between values indicates intermediate similarity or dissimilarity.

Specifically, the cosine similarity ranges from 0 to 1, when applying cosine similarity on information retrieval. This is because the frequencies must be no smaller than 0, and the angle between a pair of frequency vectors must be no larger than 90°. Cosine similarity is easy to compute and usually has a relatively accurate results. That is why researchers often take cosine similarity as the similarity measure in the field of information retrieval. For instance, we make use of cosine similarity to measure the similarities of two POIs, for recommending POIs to users, which is detailed in Chapter 4.

2.4.3 Location Identification and Recommendation

During the past few decades, studies on location identification and recommendation have appeared. [142] and [143] extract locations of interest and travel sequences based on multiple users' trajectories. However, they do not consider the semantics of locations and temporal information of users' GPS trajectories. Further, they use HITS algorithm, which may cause improper weights to be assigned to links, as discussed in the work [21]. In [21], the authors design a framework to extract semantically significant and meaningful

locations from GPS data. They rank semantic locations according to the significance of the locations. However, they do not consider temporal effects. The most significant difference of our work from the work reviewed above is that the work above does not consider personalized information (e.g., visiting history of the user for whom we make recommendations). Therefore, they will recommend the same set of locations for every user. In contrast, our work aims to provide personalized recommendation by making use of personal preferences. Hence our work reveals the periodic property of users' regular behavior and mines the semantics of a location.

[130] mines a novel Individual Life Pattern to form individual trajectory data. They use this pattern for describing and modeling periodic behaviors of mobile clients. [16] discovers more patterns of interest by using a layer of geographic semantic information.

[129] deals with the POI recommendation problem most closely relevant to ours. They adapt the collaborative filtering (CF) model for POI recommendations, so that the recommendation precision can be improved. They model the spatial influence by making use of a Bayesian CF model-based algorithm. However, they have different assumptions and significantly different techniques from our work since they emphasize the geographical influence while we extract semantic POIs from GPS data, and consolidate all the popularity, temporal, and geographical influences and propose a unified framework to address this problem. Another related work that copes with POI recommendation problems is time-aware POI recommendation ([137]). However, they focus on recommending POIs from a set of POIs rather than from users' GPS trajectories.

As far as we know, there are several works studying GPS trajectories but not for POI recommendation. The authors of [133] mine user similarity from trajectories. One year later, the same authors ([132]) study the geographical features and semantic features of GPS trajectories for prediction of users' movements. However, our work is intrinsically different from these two works. Compared to their works, our framework studies the popularity and semantic, temporal, and geographical features of the extracted POIs whereas their works only consider the geographical features and semantic features of the GPS trajectories. In addition, note that the purpose of their works is to predict users' movements

whereas our work focuses on POI recommendation for multiple users. Therefore, our work is the first one to recommend POIs to clients, by considering the semantics, popularity, temporal and geographical features of POIs together, from the raw GPS trajectory data.

2.4.4 Recommendation with Temporal Information

With the broad utilization of graphs ([124]), matrix factorization ([141]), and decision-tree ([146]), the last decades have witnessed many works on time-aware recommendation.

[124] considers the influence of people's short-term and long-term preferences on people's behaviors. For the purpose of modeling the above two types of preferences, the authors propose a graph STG, i.e., Session-based Temporal Graph. The nodes of an STG graph fall in the following three categories: *user*, *item*, and *session*. They propose an 'MS-IPF' algorithm, which stands for Multi-Source Injected Preference on the basis of the STG, for the purpose of propagating the preference from *user* and *session* nodes to candidate *item* nodes. [38] suggests that current ratings play a more important role than previous ones. In this way, the authors estimate similarities between items by decaying the weights of older ratings.

However, these studies only considers the temporal influences of different POIs. They do not take popularity or geographical information into consideration.

2.4.5 Recommendation with Geographical Information

A majority of POI recommendation works utilize the geographical influences. [25] uses a matrix factorization model to exploit geographical information by a Gaussian mixture model (GMM). [80] incorporates matrix factorization into a probabilistic model for POI recommendation. [119] filters out those POIs which are far away from the target client by utilizing the framework of personalized Pagerank. [73] also utilizes a model-based approach to sample a POI on the basis of the geographical distance to historical POIs checked-in by a client. [131] studies the different topics of different cities and an LDA-based model is proposed for recommending POIs for a specified client in a given city.

However, these studies only considers the geographical influences of different POIs. They do not take popularity or temporal information into consideration.

2.5 Cost-Optimal Route Queries in Time-Dependent Road Networks

This section reviews existing studies on route planning. We first present existing works on traditional route planning problem, followed by routing problems in static road networks, traditional time-dependent routing problem, energy-efficient path planning, and other works on path planning, respectively.

2.5.1 Traditional Route Planning Problem

Routing services have become ubiquitous in the past years. The algorithm community has conducted a great amount of work regarding speedup-techniques to the original Dijkstra's algorithm. For the purpose of accelerating the minimum-cost route calculation, a $HiTi$ (Hierarchical MulTi) graph model is established by [63], providing a novel method for structuring a topographical road map hierarchically. Another speedup technique is *Contraction Hierarchies* (CH) ([11] and [48]), which can conveniently balance the leverage between preprocess and query time. Multi-level techniques have been studied for answering shortest-path queries by [55]. [83] works with OpenStreetMap (OSM) data and provides users with real-time and exact shortest path on large networks with millions of road segments. [76] also uses OSM data and proposes an approach to extract multi-lane roads from urban road networks. [46] investigates shortest distance queries in outsourcing graph and designs methods to save computational costs while satisfying security requirements. However, these works do not consider time-dependent cost or speed, and they also do not consider travel time constraints.

2.5.2 Route Planning in Static Road Networks

For route search in static road networks, [32, 35] propose *Customizable Route Planning* (CRP). However, they do not consider time-aware scenarios. Geisberger and Vetter also address routing problems in their paper [49], however, the setting is also in static road networks. The study [122] defines, investigates and discusses a model that can handle costs of turns in route planning, but their setting is also restricted to static road networks. Based on predefined waypoints, [27] proposes a real-time route-planning algorithm, which can provide an optimal route and avoid static obstacles for one kind of autonomous vehicle. However, it focus on the smoothness, safety cost and consistency of the path, whereas in our work, we consider the total financial cost of a path, i.e., fuel cost plus toll fee. In addition, their setting is in static road networks. [106] develops the Capacity Constrained Route Planner (CCRP), which honors capacity constraints and is a generalization of shortest path algorithms. [106] also proposes algorithms for exploiting the spatial structures of road networks to speedup routing for large networks. However, their problem setting is different from ours since they neither consider fuel consumption or toll fees, nor consider the time constraint.

2.5.3 Traditional Time-Depenent Routing Problem

There exist some studies on time-dependent route planning ([104, 37, 23]). In [104], the authors use graphs of multiple levels, for timetable information in train systems, but they do not take energy consumption into account. [37] defines a time-dependent shortest-path (TDSP) problem and proposes a solution to answer the LTT (least total travel time) query. [23] corrects and extends a few state-of-the-art dynamic shortest-path-tree algorithms to handle multiple edge weight updates. All these studies assume that the departure time of a node n_i is always the same as the arrival time of n_i. However, this property does not hold in our setting as we allow waiting at any node. Therefore, these solutions cannot solve COTER.

2.5.4 Energy-Efficient Path Planning

As mentioned in Chapter 1, when answering route queries for moving users, the key point is to find a path that minimizes travel cost. Some studies consider the energy consumption by a vehicle as the travel cost. For example, [115] and [36] focus on energy consumption when planning routes in static road networks.

In [12], the authors consider the speed-energy tradeoff for electrical vehicles and use multi-criteria optimization to obtain Pareto sets of routes that trade energy consumption for speed. In [53], they not only make use of variation of the routes to save energy but also allow variation of driving speed along the route to achieve energy savings. The two works above aim to minimize energy consumption by considering speeds, but they do not take time-dependent toll fee of an edge into account. In addition, the problems addressed by these two works neither allow waiting at a node, nor have time constraints, e.g., the departure time should be after t_d and the arrival time should be before t_a.

2.5.5 Other Works on Path Planning

The study [70] considers multiple preferences such as *driving time*, *distance*, etc, to process skyline queries in a static road network. The paper [20] defines the keyword-aware optimal route query (KOR), with the aim of finding an optimal route covering all the user-specified keywords, satisfying a specified budget constraint and inducing an optimal objective score. But their setting is restricted to static road networks. The problem of KSP (computing the top K-Shortest Paths in a network) as well as NSP (Near-Shortest Paths problem), has a long history. For example, [85] studies KSP and NSP by proposing algorithms which are faster than other algorithms in literature at that time by more than one order of magnitude. [3] proposes a K^* algorithm, for solving KSP in a directed weighted graph. [41] proposes a "simplest" path algorithm and deduces that the length of a simplest path is just 16% longer than that of the corresponding shortest path on average. However, the problem setting and objectives of KSP or NSP are obviously different from ours. [58] considers safety, costs and security when planning optimal routes for HAZMATs (trucks that carry hazardous materials) in road networks, but this work does not

have time constraint and has different problem setting from ours.

2.5.6 Comparison with Other Cost-Optimal Routing Problems

The most relevant studies, [18], [128], and [10] solve cost-optimal problems under travel time constraints. Compared with the first two works ([18, 128]), they also allow waiting at a node and use dynamic programming to calculate the optimal cost, However, (i) they consider neither multiple costs, nor maximum speed constraints, and therefore the problem proposed in our work is novel; (ii) they either disallow waiting at any node or allow waiting at all nodes, but we only allow waiting at some nodes and at other nodes waiting is disallowed; (iii) our toll fee function could be any arbitrary single-valued function whereas [128] only allows the toll fee to be a piecewise constant function; (iii) our approach employs the technique of nonlinear programming optimization whereas [18, 128] do not; (iv) our paper can compute the optimal cost faster than [128], since our algorithm first computes the feasible arrival time interval and gains a topological sorting for the candidate nodes, which prunes much search space for recursively computing the optimal cost. Compared with the third work ([10]), though [10] also adopts the technique of nonlinear programming optimization, (i) it does not allow waiting at any node; (ii) it only considers the total energy cost (fuel and electricity) and does not consider other kinds of cost, e.g., time-dependent toll fee, at all.

Chapter 3

Proximity Detection in Road Networks

This chapter solves the problem of proximity detection in road networks. This chapter is structured as follows. Section 3.1 discusses the problem setting of proximity detection, and gives some definitions to model the road network. Section 3.2 proposes a fixed-radius mobile detection solution to address this problem. Section 3.3 proposes a self-adjustment solution to address this problem. Section 3.4 discusses methods for server-side computational cost optimization. Section 3.5 presents the experimental results of our methods. Finally, Section 3.6 concludes this chapter.

3.1 Definitions and Problem Setting

We first give some definitions to model road networks, and then present the problem setting.

Definition 1 ***junction.*** *A junction is an intersection of two or more different line segments.* ♡

As shown in Fig. 3.1(a), J_2, J_5 are typical junctions, and J_3, J_6, J_7, J_8 are degenerated junctions.

Definition 2 ***edge.*** *An edge is a line segment between two adjacent junctions.* ♡

As shown in Fig. 3.1(a), $\langle J_1, J_2\rangle$, $\langle J_1, J_4\rangle$, $\langle J_4, J_5\rangle$, $\langle J_2, J_5\rangle$, $\langle J_5, J_6\rangle$ are all edges.

Definition 3 ***road network.*** *Let $J = \{junction\}$, $E = \{edge\}$, a road network $G = (J, E)$.* ♡

For example, Fig. 3.1(a) depicts a simple road network.

Definition 4 ***network point.*** *A network point is a point on the edges of a road network.*♡

For example, P_1, Q_1, P_2, Q_2, J_2 and J_6 in Fig. 3.1(a) are all network points.

Definition 5 ***network distance.*** *Given two locations (network points) P, P' in the road network, their network distance is given by*

$$D(P, P') = \min_{\substack{a\in\{s,t\},\\ b\in\{s',t'\}}} (D(P, J_a) + D(J_a, J_b) + D(J_b, P')) \quad (3.1)$$

♡

The network distance between two network points is equal to the shortest-path length of the two points. For example, in Fig. 3.1(a), the Euclidean distance between P_1 and Q_1 is denoted as $|P_1Q_1|$, while the network distance between them is:

$$D(P_1, Q_1) = \min_{\substack{a\in\{2,3\},\\ b\in\{5,6\}}} (D(P_1, J_a) + D(j_a, J_b) + D(J_b, Q_1)).$$

Definition 6 ***line segment.*** *A line segment is a segment between two network points on the same edge. The first network point is the starting point and the second network point is the ending point.* ♡

As shown in Fig. 3.1(a), $\overline{P_1P_2}$, $\overline{Q_1Q_2}$ are line segments.

Definition 7 ***mobile region.*** *A mobile region of a client is a line segment on one edge or several line segments on several edges. More formally, a mobile region is a tree composed of a sequence of line segments. The network distance from the root of this tree, to each of its leaves is equal. We represent a mobile region of an object O_m at time t by $R_m(t)$.* ♡

Suppose $(A_1, A_2, \ldots, A_p)$ are vertices of $R_m(t)$, and $(B_1, B_2, \ldots, B_q)$ are vertices of $R_n(t)$, then $R_m(t)$ and $R_n(t)$ have $(p-1)$ edges and $(q-1)$ edges, respectively.

$$R_m(t) = \bigcup_{1\le i\le p-1} e_i^m \ , \quad R_n(t) = \bigcup_{1\le j\le q-1} e_j^n \ . \quad (3.2)$$

where, e_i^m and e_j^n represent an arbitrary edge of $R_m(t)$ and $R_n(t)$. $e_i^m = \overline{A_{i_1}A_{i_2}}, e_i^n = \overline{B_{j_1}B_{j_2}}$. As shown in Fig. 3.1(b), the mobile region of O_1 at time t $R_1(t)$ = $\{\overline{P_1P_2}\}$ and the mobile region of O_2 at time t $R_2(t)$ = $\{\overline{Q_1J_5}, \overline{J_5Q_2}, \overline{J_5Q_{2'}}, \overline{J_5Q_{2''}}\}$. We shall explain mobile regions in detail in Section 3.2.1.

Definition 8 ***proximity detection in road network.*** *We are given a set of moving objects O on a road network G, each with different speed at different time, their friendship, and*

a network distance threshold for every pair of friends. Our task is to design efficient algorithms with a low communication cost, to find each pair of friends whose network distance does not exceed the distance threshold, based on the client-server architecture.♡

We assume that every moving client has a positioning device. The server is equipped with a map of the road network and knows the length of each edge, and the coordinates of each junction of the road network. The server detects whether each friend pair is within proximity periodically (e.g., every second).

Notations. Table 3.1 lists the symbols used in Chapter 3.

Table 3.1: Notations.

Notation	Meaning
$\langle J_s, J_t \rangle$	edge (route) connecting junctions J_s and J_t
$\overline{PQ}$	line segment $\overline{PQ}$
$\|PQ\|$	Euclidean distance between P and Q
$D(P, Q)$	network distance between P and Q
$R_m(t)$	mobile region of Object O_m at time t
$d_{min}(R_m(t), R_n(t))$	Euclidean distance based lower bound of $D(R_m(t), R_n(t))$
$D_{min}(R_m(t), R_n(t))$	network distance based lower bound of $D(R_m(t), R_n(t))$
$D_{max}(R_m(t), R_n(t))$	network distance based upper bound of $D(R_m(t), R_n(t))$

3.2 Fixed-Radius Mobile Detection (FRMD)

In this section, we present a basic client-server solution with a fixed-radius mobile region (FRMD) for each client. We first present the mobile regions used in our method, and then propose three pruning lemmas which make use of two lower bounds and one upper bound of the distance between the mobile regions of two clients, for the server to prune unnecessary probing messages. We subsequently present the algorithms at the client and server side, and finally analyze the communication cost model of this FRMD method.

3.2.1 Mobile Regions

We define a mobile region for each client. It is designed in such a way that unless a client moves outside its mobile region, no update is needed. Unlike the mobile region proposed

in [134] which is a circle, our mobile region is a line segment or several line segments along the road network.

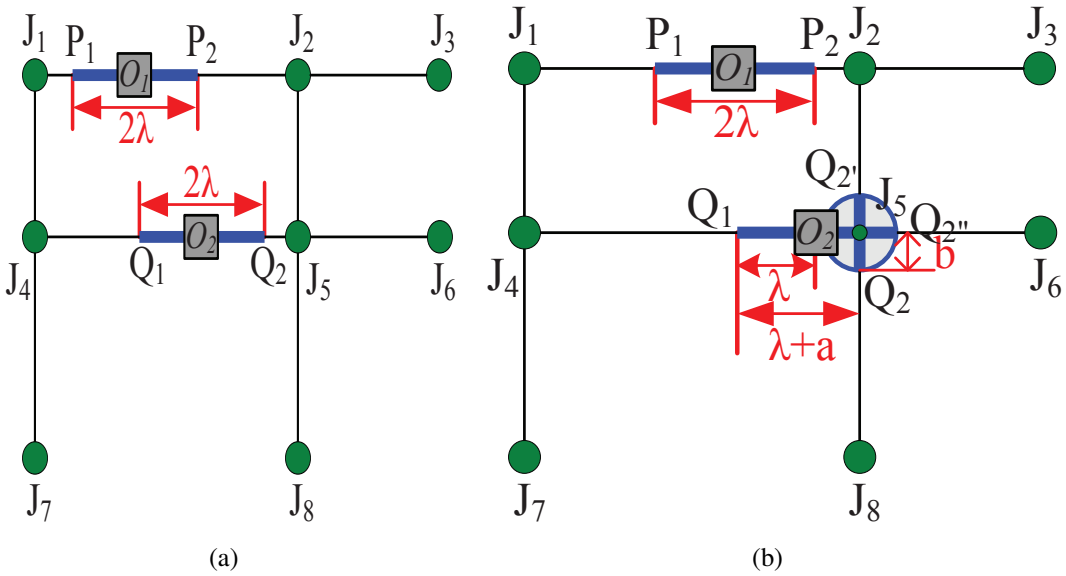


Figure 3.1: Fixed-radius mobile region. (a) mobile region at time T_1 ; (b) mobile regions at time T_2=T_1+ΔT.

Suppose λ denotes the fixed radius of the mobile region of each client. The position of the center of the mobile region at T_{last}+ΔT is calculated as follows:

$$P_{(T_{last}+\Delta T)} = P_{T_{last}} + v_{last} \cdot \Delta T \tag{3.3}$$

where, T_{last} is the time of last update, $P_{T_{last}}$ is the position of the client at time T_{last}, and v_{last} is the speed at the time of last update. We give an example to explain the mobile regions.

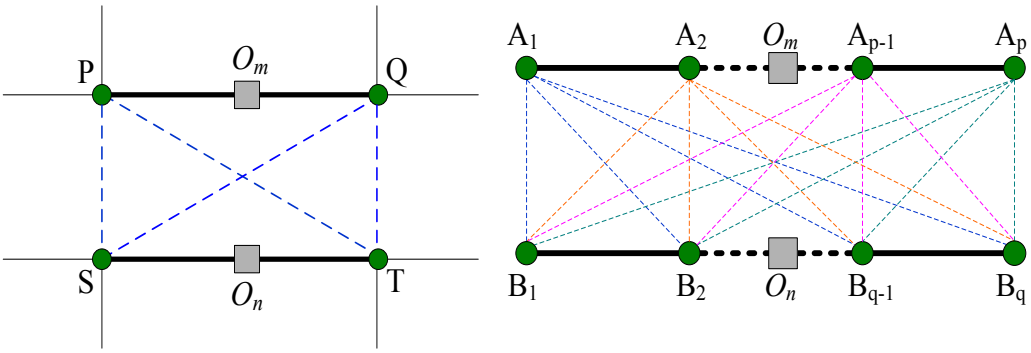
As shown in Fig. 3.1, O_1 and O_2 are two clients moving along edges $\langle J_1, J_2\rangle$ and $\langle J_4, J_5\rangle$. In Fig. 3.1(a), suppose the two clients O_1 and O_2 have just reported their position update messages to the server at time T_1, and the line segments $\overline{P_1P_2}$ and $\overline{Q_1Q_2}$ with length 2λ in BLUE are the mobile regions of O_1 and O_2, with O_1 and O_2 being the midpoints of $\overline{P_1P_2}$ and $\overline{Q_1Q_2}$, respectively. In Fig. 3.1(b), at time T_2, O_1 and O_2 send position update messages to the server again, then line segment $\overline{P_1P_2}$ with length 2λ is the new mobile region of O_1 with the midpoint O_1. As O_2 approaches J_5, its mobile region is no longer completely within the edge $\langle J_4, J_5\rangle$. Suppose O_2 is a-unit distance

away from J_5, where $a < \lambda$, then the leftmost point Q_1 of its mobile region is (λ + a) units away from J_5. Here, O_2 has three different directions to move along when it arrives at junction J_5, then our method includes edges $\overline{J_5Q_2}$, $\overline{J_5Q_{2'}}$, $\overline{J_5Q_{2''}}$ into its mobile region. Suppose $|\overline{J_5Q_2}| = |\overline{J_5Q_{2'}}| = |\overline{J_5Q_{2''}}| = b$, then b = λ - a.

3.2.2 Pruning Lemmas

The server does not have to probe each client even if the client does not send an update message in one epoch. We propose three pruning lemmas that make use of the lower bounds and upper bounds of the network distance between two clients' mobile regions to reduce unnecessary probing messages.

3.2.2.1 Euclidean distance based Lower bound pruning



(a) (b)

Figure 3.2: Two moving objects and their mobile regions

Theorem 3.1. *Given two mobile regions $R_m(t)$ and $R_n(t)$, then the following formula holds.*

$$d_{min}(R_m(t), R_n(t)) = \min_{1 \le i \le p, 1 \le j \le q} |A_iB_j|, \tag{3.4}$$

where, $(A_1, A_2, \ldots, A_p)$ are vertices of $R_m(t)$, and $(B_1, B_2, \ldots, B_q)$ are vertices of $R_n(t)$.♠

PROOF. From Eq. 3.2, we obtain,

$$\begin{aligned} & d_{min}(R_m(t), R_n(t)) \\ = & \min_{1 \le i \le p-1, 1 \le j \le q-1} d_{min}(e_i^m, e_j^n) \\ = & \min_{1 \le i \le p, 1 \le j \le q} |A_iB_j| \end{aligned}$$

■

For example, as shown in Fig. 3.2(a), $\forall O_m \in \overline{PQ}$, $\forall O_n \in \overline{ST}$, then,

$$\begin{aligned} & D(O_m, O_n) \\ = \; & \min\{D(O_m, P) + D(P, S) + D(S, O_n), \\ & \quad D(O_m, P) + D(P, T) + D(T, O_n), \\ & \quad D(O_m, Q) + D(Q, S) + D(S, O_n), \\ & \quad D(O_m, Q) + D(Q, T) + D(T, O_n)\} \\ \geq \; & \min\{D(P, S), D(P, T), D(Q, S), D(Q, T)\} \\ \geq \; & \min\{|\ PS\ |, |\ PT\ |, |\ QS\ |, |\ QT\ |\} \end{aligned}$$

Therefore, $\min\{|\ PS\ |, |\ PT\ |, |\ QS\ |, |\ QT\ |\}$ is the lower bound of $D(\overline{PQ}, \overline{ST})$.

Lemma 1. ***Unqualified-pair pruning I.*** *If $d_{min}(R_m(t), R_n(t)) > \epsilon_{m,n}$, then the exact network distance between O_m and O_n must be larger than $\epsilon_{m,n}$, therefore this friend pair should be pruned.* ♣

PROOF. Since $d_{min}(R_m(t), R_n(t))$ is the lower bound of the distance between $R_m(t)$ and $R_n(t)$, thus $d_{min}(R_m(t), R_n(t))$ is also the lower bound of the network distance between O_m and O_n. Obviously, $D(O_m, O_N) \geq d_{min}(R_m(t), R_n(t)) > \epsilon_{m,n}$. Therefore Lemma 1 holds. ■

Discussion. As $d_{min}(R_m(t), R_n(t))$ is calculated by using the metric of Euclidean distance, Lemma 1 is also called an Euclidean distance based lower bound pruning lemma.

3.2.2.2 Network-distance based lower bound and upper bound pruning

Theorem 3.2. *Given two clients O_m, O_n as well as their mobile regions $R_m(t)$ and $R_n(t)$, as depicted in Fig. 3.2(b), we obtain,*

$$\begin{aligned} D_{min}(R_m(t), R_n(t)) &\leq D(R_m(t), R_n(t)) \\ &\leq D_{max}(R_m(t), R_n(t)) \end{aligned} \tag{3.5}$$

where

$$D_{min}(R_m(t), R_n(t)) = \min_{1\le i\le p, 1\le j\le q} D(A_i, B_j) \tag{3.6}$$

$$\begin{aligned} & D_{max}(R_m(t), R_n(t)) \\ = & \max_{\substack{1\le i\le p-1 \\ 1\le j\le q-1}} \{\min(D(A_{i_1}, B_{j_1}), D(A_{i_1}, B_{j_2}), D(A_{i_2}, B_{j_1}), \\ & D(A_{i_2}, B_{j_2})) + |e_i^m| + |e_j^n|\}. \end{aligned} \tag{3.7}$$

♠

PROOF. $\forall P \in \overline{A_{i_1}A_{i_2}}, \forall P' \in \overline{B_{j_1}B_{j_2}}$,

$$\begin{aligned} D(P, P') = \min\{ & \\ & D(A_{i_1}, B_{j_1}) + \mid A_{i_1}P \mid + \mid B_{j_1}P' \mid, \\ & D(A_{i_2}, B_{j_1}) + \mid A_{i_2}P \mid + \mid B_{j_1}P' \mid, \\ & D(A_{i_1}, B_{j_2}) + \mid A_{i_1}P \mid + \mid B_{j_2}P' \mid, \\ & D(A_{i_2}, B_{j_2}) + \mid A_{i_2}P \mid + \mid B_{j_2}P' \mid \\ & \} \end{aligned}$$

Observe that $\forall i \in \{1, 2, \ldots, p-1\}$, $\forall j \in \{1, 2, \ldots, q-1\}$, $|A_{i_1}P| \ge 0$, $|B_{j_1}P'| \ge 0$, $|A_{i_2}P| \ge 0$, $|B_{j_2}P'| \ge 0$ always hold, so it is obvious that

$$D(R_m(t), R_n(t)) \ge \min_{1\le i\le p, 1\le j\le q} D(A_i, B_j).$$

Meanwhile, observe that for $1 \le i \le p-1$, $\forall P \in \overline{A_{i_1}A_{i_2}}$, $|A_{i_1}P| \le |\overline{A_{i_1}A_{i_2}}|$, $|A_{i_2}P| \le |\overline{A_{i_1}A_{i_2}}|$ always hold, and for $1 \le j \le q-1$, $\forall P' \in \overline{B_{j_1}B_{j_2}}$, $|B_{j_1}P'| \le |\overline{B_{j_1}B_{j_2}}|$, $|B_{j_2}P'| \le |\overline{B_{j_1}B_{j_2}}|$ always hold. Therefore, we obtain

$$\begin{aligned} & D(R_m(t), R_n(t)) \\ \le & \max_{\substack{1\le i\le p-1 \\ 1\le j\le q-1}} \{\min(D(A_{i_1}, B_{j_1}), D(A_{i_1}, B_{j_2}), D(A_{i_2}, B_{j_1}), \\ & D(A_{i_2}, B_{j_2})) + |\overline{A_{i_1}A_{i_2}}| + |\overline{B_{j_1}B_{j_2}}|\}. \end{aligned}$$

As a result, Theorem 3.2 holds. ■

Lemma 2. ***Unqualified-pair pruning II.***

If $D_{min}(R_m(t), R_n(t)) > \epsilon_{m,n}$, then the exact network distance between O_m and O_n must be larger than the proximity threshold $\epsilon_{m,n}$, then this pair of moving objects must be

pruned. ♣

PROOF. From Theorem 3.2, we know that $D(R_m(t), R_n(t)) \geq D_{min}(R_m(t), R_n(t))$. Since O_m and O_n locate inside $R_m(t)$ and $R_n(t)$, thus $D(O_m, O_n) \geq D_{min}(R_m(t), R_n(t))$.

If $D_{min}(R_m(t), R_n(t)) > \epsilon_{m,n}$, then $D(O_m, O_n) > \epsilon_{m,n}$. That is to say, the network distance between O_m and O_n is larger than $\epsilon_{m,n}$, and therefore Lemma 2 holds. ■

Lemma 3. ***Qualified-pair pruning.***

If $D_{max}(R_m(t), R_n(t)) \leq \epsilon_{m,n}$, then the exact network distance between O_m and O_n must be at most the same as the proximity threshold $\epsilon_{m,n}$, then this pair must be within proximity. ♣

PROOF. This proof is analogous to that of Lemma 2. We know that $D(R_m(t), R_n(t)) \leq D_{max}\ (R_m(t), R_n(t))$ from Theorem 3.2 and because O_m and O_n locate inside $R_m(t)$ and $R_n(t)$, thus $D(O_m, O_n) \leq D_{max}(R_m(t), R_n(t))$. If $D_{max}(R_m(t), R_n(t)) \leq \epsilon_{m,n}$, then $D(O_m, O_n) \leq \epsilon_{m,n}$ holds. That is to say, the exact network distance between O_m and O_n is no larger than the threshold $\epsilon_{m,n}$ and this pair is within proximity. Therefore Lemma 3 holds. ■

Discussion. As $D_{min}(R_m(t), R_n(t))$ and $D_{max}(R_m(t), R_n(t))$ are computed by using the metric of network distance, Lemma 2 and Lemma 3 are also called network-distance based lower bound and upper bound pruning lemmas.

3.2.3 Server-Side and Client-Side Algorithms

Based on the concept of mobile regions for each client and the pruning lemmas presented above, we propose the algorithms for the client and server respectively.

The server-side algorithm is described in Algorithm 1.

When the Euclidean distance and the network distance based lower bounds between two friends are not larger than ϵ and the network distance based upper bound between two friends is smaller than or equal to ϵ, the server needs to notify the two friends. Otherwise, the server probes the client that has not updated the server at the current epoch, and computes the exact network distance between the two clients and then notify them if the distance is smaller than or equal to ϵ.

The client-side algorithm is as follows.

Algorithm 1: Server-side algorithms of FRMD

```
1  for (t = initTS; t ≤ MaxTS; t += ΔT) do
2      server.receiveUpdateFromClient(speed, loc);
3      for each friend pair O_m and O_n do
4          if d_min(R_m(t), R_n(t)) > ε_{m,n} then
5              continue;
6          end
7          if D_min(R_m(t), R_n(t)) > ε_{m,n} then
8              continue;
9          end
10         if D_max(R_m(t), R_n(t)) ≤ ε_{m,n} then
11             server.notify(O_m, O_n);
12         end
13         else
14             if client.notUpdate(O_m) then
15                 server.probe(O_m);
16             end
17             if client.notUpdate(O_n) then
18                 server.probe(O_n);
19             end
20             if D(O_m, O_n) ≤ ε_{m,n} then
21                 server.notify(O_m, O_n);
22             end
23         end
24     end
25 end
```

(i) When moving beyond its mobile region, the client updates the server.

(ii) When probed by the server, the client updates the server.

3.2.4 Communication Cost Analysis of FRMD

Now we analyze the total communication cost of our solution induced per epoch. First, suppose there are N clients in total and each client has an average of m friends. Let ΔT denote the length of one epoch. All of the friend pairs share the same proximity threshold. The total communication cost CC_{total} can be decomposed into three kinds of cost: (i) the update cost CC_{update}, which measures how many messages issued by the clients to the server; (ii) the probing cost CC_{probe}, which is induced by the server which probes the clients regarding the exact positions in the "refinement step", together with the associated invoked update messages replied by the clients; (iii) The proximity notification

cost CC_{notify}, which counts how many notification messages delivered by the server to clients which are within proximity. We disregart CC_{notify} since it has nothing to do with the radius of mobile regions.

3.2.4.1 Cost of update

Assume the actual average speed in this time interval is v' and the client measures its location every ΔT time units, then we obtain the number of updates for it in this time interval is $\min\{\frac{v' \cdot \Delta T}{\lambda}, 1\}$.

$$p_{update} = \min\{\frac{v' \cdot \Delta T}{\lambda}, 1\}$$

Therefore CC_{update} for N drivers can be expressed as follows.

$$CC_{update} = N \cdot \min\{\frac{v' \cdot \Delta T}{\lambda}, 1\} \tag{3.8}$$

From Eq. 3.8, we see we should maximize λ to minimize CC_{update}.

3.2.4.2 Cost of probing

As discussed in Section 3.2.3, the server first checks whether a pair of friends satisfies Lemma 1, then Lemma 2 and Lemma 3 in sequence. The refinement step is carried out by the server only if all the three lemmas are not satisfied by a friend pair.

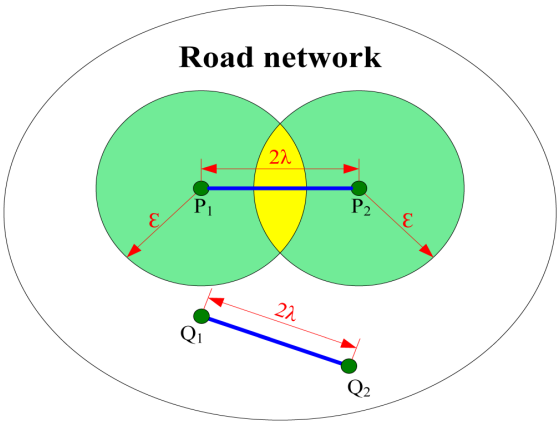


Figure 3.3: The area $\overline{Q_1Q_2}$ may lie in to satisfy Lemma 1.

Probability of not satisfying Lemma 1. Let E denote the total number of edges in G and S denote the entire area of the road network G. Assume that the edges are randomly

deployed in the road network area according to a two-dimensional Poisson process with average edge density ξ. We sample the edges from the road network to calculate the average edge density. In such cases, the position of each edge is independent.

There are $\binom{E}{1}$ different choices of choosing one edge for line segment $\overline{P_1P_2}$ to lie on. As long as the position of segment $\overline{P_1P_2}$ has been determined, if Lemma 1 is satisfied, since the minimum Euclidean distance is larger than ϵ, Q_1 and Q_2 must lie inside the road network area but outside the two green circles (Fig. 3.3). The area in which $\overline{Q_1Q_2}$ may lie in order to satisfy Lemma 1 can be calculated by Eq. 3.9.

$$Area(Q_1, Q_2) = S - 2 \cdot \pi \cdot \epsilon^2 + Area_{common} \tag{3.9}$$

where, $Area_{common}$ is the area of overlapping region (yellow part in Fig. 3.3) of the two circles.

$$Area_{common} = 2 \cdot [\arccos(\min\{\frac{\lambda}{\epsilon}, 1\}) - \lambda \cdot \sqrt{\lambda^2 - \epsilon^2}]$$

Therefore, there are a total of $\binom{E}{1} \cdot \xi \cdot Area(Q_1, Q_2)$ different choices for selecting two edges on which the two line segments $\overline{P_1P_2}$ and $\overline{Q_1Q_2}$ lie according to Lemma 1. Since the total number of choices in selecting two edges from the whole network is E^2, the probability for the two mobile regions to satisfy Lemma 1 is obtained as follows.

$$p_{Lemma1} = \frac{\binom{E}{1} \cdot \xi \cdot Area(Q_1, Q_2)}{E^2} \tag{3.10}$$

We can derive the probability of not satisfying Lemma 1 as Eq. 3.11.

$$\begin{aligned} p_1(\lambda) &= 1 - p_{Lemma1} = 1 - \frac{\binom{E}{1} \cdot \xi \cdot Area(Q_1, Q_2)}{E^2} \\ &= 1 - \xi \cdot \frac{\{S - 2\pi\epsilon^2 + 2[\arccos(\min\{\frac{\lambda}{\epsilon}, 1\})\epsilon^2 - \lambda\sqrt{\epsilon^2 - \lambda^2}]\}}{E} \end{aligned} \tag{3.11}$$

Our objective is to minimize the probability of probing therefore we need to first minimize

this $p_1(\lambda)$. We calculate its derivative.

$$\begin{aligned} & p_1'(\lambda) \\ & = -2 \cdot \frac{\xi}{E} \cdot [-\frac{\frac{1}{\epsilon} \cdot \epsilon^2}{\sqrt{1-(\frac{\xi}{\epsilon})^2}} - \frac{\epsilon^2 - 2 \cdot \lambda^2}{\sqrt{\epsilon^2 - \lambda^2}}] \\ & = -2 \cdot \frac{\xi}{E} \cdot (-2 \cdot \sqrt{\epsilon^2 - \lambda^2}) \\ & = 4 \cdot \frac{\xi}{E} \cdot \sqrt{\epsilon^2 - \lambda^2} \\ & \geq 0 \end{aligned} \tag{3.12}$$

From Inequality 3.12, we know $p_1(\lambda)$ becomes smaller as λ gets smaller. Therefore we need to minimize λ to minimize $p_1(\lambda)$.

Probability of not satisfying Lemma 2 and Lemma 3. Since Lemma 2 and Lemma 3 have very small relationship with λ, we disregard this part. Thus we obtain the probability of the refinement step is: $p_{refine} \approx p_1(\lambda)$. The total probing cost is:

$$\begin{aligned} & CC_{probe} \\ & = N \cdot (1 - p_{update}) \cdot [1 - (1 - p_{refine})^m] \cdot 2 \\ & = 2 \cdot N \cdot (1 - \min\{\frac{v' \cdot \Delta T}{\lambda}, 1\}) \cdot [1 - \xi \cdot \\ & \frac{\{S - 2\pi\epsilon^2 + 2[\arccos(\min\{\frac{\lambda}{\epsilon}, 1\})\epsilon^2 - \lambda\sqrt{\epsilon^2 - \lambda^2}]\}}{E})^m] \end{aligned} \tag{3.13}$$

From Eq. 3.13 and the above discussion, we know CC_{probe} can be minimized if λ is minimized.

Discussion. In summary, we see that a large λ induces a small CC_{update} while a small λ induces a small CC_{probe}, so there must exist an optimal λ that induces a small sum of CC_{update} and CC_{probe}.

In order to address this problem as well as reducing the total communication cost, developing efficient methods to minimize the communication cost is crucial.

3.3 Methods with Self-Adjustment

In this section, we present three self-adjustment methods to automatically adjust the size of a mobile region. We first give the RMD_{RN}/CMD_{RN} method and then propose the RRMD method.

Two operations are introduced to adjust the mobile regions to control the update cost and probing cost when necessary. Here, there is a tuning parameter α to adjust the length of our mobile regions.

(i) Expansion: The radius of a mobile region is multiplied by α. Thus the total length of the mobile region has expanded.

(ii) Contraction: The radius of a mobile region is divided by α. Thus the total length of the mobile region has contracted.

Note that we use division and multiplication instead of subtraction and addition, since division and multiplication can contract and expand a mobile region to a larger extent than subtraction and addition.

Adjust rule:

(i) When the probability of probing is too high, that is, the radius is too large, we should conduct the contraction operation.

(ii) When the probability of update is too high, that is, the radius is too small, we should conduct the expansion operation.

3.3.1 RMD_{RN}/CMD_{RN} Method

We adopt the idea of self-tuning methods, Reactive Mobile Detection (RMD) and Cost-based Mobile Detection (CMD) designed for proximity detection in Euclidean space in [134] and propose self-adjustment methods RMD_{RN} (Reactive Mobile Detection for Road Network) and CMD_{RN} (Cost-based Mobile Detection for Road Network) by taking road network constraints into consideration.

For RMD_{RN} method, when a client is about to transmit an update to the server, it conducts an expansion operation to reduce future update possibility, which follows the

idea of the RMD method [134]. When a client receives a probing message from the server, it conducts a contraction operation.

For CMD_{RN} method, each client maintains two counters *UpdateCount* and *ProbeCount* which represent the probabilities of update and probing messages, following the idea of the RMD method [134]. When a client finds it is outside its mobile region, its *UpdateCount* increases by 1. If *UpdateCount* $>$ *ProbeCount*, it conducts an expansion. When a client receives a probing message from the server, its *ProbeCount* increases by 2. If *ProbeCount* $>$ *UpdateCount*, it performs a contraction.

We have conducted experiments to study the behavior of these two methods and found that these two methods can reduce the total communication cost to some extent.

3.3.2 Radius-Based Reactive Mobile Detection Method (RRMD)

Our RRMD method is designed on the basis of the FRMD and RMD_{RN} methods. In addition to the tuning parameter α, we introduce a second parameter β. In order to make our RRMD method more efficient, we set up the relationship between the initial radius, optimal radius of FRMD and β. The goal is to take advantage of the RMD_{RN} method and make use of the relation of initial radii and the optimal radius and thus incur a lower total communication cost.

It is intuitive that no matter how much the initial radius is, the optimal radius in our FRMD method that minimizes the total communication cost fluctuates slightly around a fixed value. Without loss of generality, the relationship between the initial radii $\lambda_{initial}$ and the optimal radius $\lambda_{optimal}$ together with the adjustment factor β can be described as follows.

$$\beta = \frac{\lambda_{optimal}}{\lambda_{initial}} \tag{3.14}$$

The value of $\lambda_{optimal}$ is 20 by default. Then the tuning factor β can be calculated according to Eq. 3.14. The update and probing related method of RRMD is described in Algorithms 2 and 3. ts is the current time stamp and id is the ID of this client.

In Algorithm 2, when a client is outside its mobile region, if β is no more than 1.0 or β is no more than α, the client applies expansion to its mobile region which multiplies

Algorithm 2: update related method of RRMD

```
1  if client.IsOutsideMobileRegion(ts, id) then
2      if β > 1 and β > α then
3          r' = client.expandMobileRegion(β, r);
4          server.computeMobileRegionForObject(ts, id, r');
5      end
6      else
7          r' = client.expandMobileRegion(α, r);
           server.computeMobileRegionForObject(ts, id, r');
8      end
9      if client.IsOutsideMobileRegion(ts, id) then
10         client.updateToServer(ts, id);
11         costOfUpdate++;
12         if server.notReceivedUpdateFrom[id] then
13             server.receivedUpdateFrom[id] = true;
14         end
15     end
16 end
```

its $\lambda_{initial}$ by α. Otherwise, the client applies expansion which multiplies its $\lambda_{initial}$ by β. Therefore the radius of the mobile region always oscillates around the optimal radius to avoid it from getting too small or too large. After this step the client checks whether it is still outside its new mobile region, if so the client updates the server with its location.

In Algorithm 3, when a client receives a probing message from the server, it applies a contraction operation to its mobile region, which divides its radius $\lambda_{initial}$ by β when β is larger or equal to α, and divides its $\lambda_{initial}$ by α when β is smaller than α.

Algorithm 3: probe related method of RRMD

```
1  if server.notReceivedUpdateFrom[id] then
2      server.probeClient(id, ts);
3      if β < α then
4          r' = client.contractMobileRegion(α, r);
5          server.computeMobileRegionForObject(ts, id, r');
6      end
7      else
8          r' = client.contractMobileRegion(β, r);
9          server.computeMobileRegionForObject(ts, id, r');
10     end
11     client.updateToServer(ts, id);
12 end
```

The advantage of our RRMD method is that it automatically uses different β instead of one fixed α to tune the radii of mobile regions. It is efficient and scalable to various radii and reduces communication cost greatly. Detailed experiments are explained in Section 3.5.

3.4 Server-Side Computational Cost Optimization

This section presents server-side cost optimization algorithms, covering proximity notification optimization, junction-to-junction network distance computation, and trigger time technique.

3.4.1 Proximity Notification Optimization

In our original server-side algorithm described in Section 3.2.3, if a friend pair is within a proximity threshold for a long time interval, for example, from time stamp T_1 = 3 to T_2 = 60, the server has to send notification messages to them at every time stamp during the time interval [3, 60]. Thus a high notification cost is incurred. Hence we use the proximity notification optimization technique to reduce the number of notification messages.

Let S, S' denote the result set in current and previous epochs. The numbers of friend pairs in S and S' are n and n', respectively.

(i) Suppose the friend pair belongs to S but not S', then the server sends a true-status message to this pair.

(ii) Suppose the friend pair belongs to S' but not S, then the server sends a false-status message to this pair.

To speed up the search, we use a red-black tree (a kind of binary search tree) with unique friend pair IDs as search keys. In this way, the time complexity for searching the friend pairs of S in the previous result set S' is $O(n \log n')$. The space complexity of this technique is $O(n+n')$.

3.4.2 Junction-to-Junction Network Distance Computation

Junction-to-junction network distance is the shortest length between two junctions on a road network. The server needs to compute $D_{max}(R_m(t), R_n(t))$ and $D_{min}(R_m(t), R_n(t))$ between two mobile regions in Lemma 2 and Lemma 3. According to Eq. 3.6 and 3.7, we first calculate the network distance between each vertex A_i of $R_m(t)$ and each vertex B_j of $R_n(t)$. We compute $D(A_i, B_j)$ by substituting the related junction-to-junction network distance into Eq. 3.1. We implement two methods of computing junction-to-junction network distance.

- **Method 1: Direct junction-to-junction network distance precomputation.** This method precomputes the shortest path between any two junctions by using the classical Dijkstra's algorithm. Thus, to calculate $D_{max}(R_m(t), R_n(t))$ and D_{min} $(R_m(t),\ R_n(t))$, the server computes $D(A_i, B_j)$ by directly substituting the corresponding precomputed junction-to-junction network distance into Eq. 3.1. The advantage of Method 1 is that it saves computational cost during execution by precomputation.

- **Method 2: "SKETCH" and Bourgain algorithm**. In [29], for every node in the graph, a small "sketch" is computed and stored. The precomputed values will be looked up and a simple computation will be performed to estimate the distance at query time. We calculate the lower bound and upper bound of the network distance between every two junctions. Thus the server uses the lower and upper bound of a junction-to-junction distance to approximately compute $D_{max}(R_m(t), R_n(t))$ and $D_{min}(R_m(t), R_n(t))$. The advantage of this method is that it computes the lower and upper bound of the junction-to-junction network distance much faster in large road networks. In addition, this method saves much memory.

3.4.3 Trigger Time Technique

The server needs to check each friend pair $\langle O_m, O_n \rangle$ in each epoch. To ease the computation burden at the server side, a trigger time technique is applied to filter out more friend

pairs.

3.4.3.1 Trigger time

Inspired by the trigger time concept in [61] and [134], we propose our trigger time. The trigger time $\omega(O_m, O_n)$ for objects O_m and O_n is defined as the **earliest** time t ($t \geq t_{cur}$) when the minimum Euclidean distance between $R_m(t)$ and $R_n(t)$ is within ϵ. Only when the minimum Euclidean distance is smaller than ϵ, the network distance between them is possibly within ϵ. The server does not need to handle the pair $\langle O_m, O_n \rangle$ until the time $\omega(O_m, O_n)$.

$$\omega(O_m, O_n) = \min\{t | t \geq t_{cur} \wedge |(R_m(t), R_n(t))| \leq \epsilon\}$$

Trigger time computation. Suppose two moving objects move toward each other on a straight line, in this way the Euclidean distance between them becomes smallest. Suppose their speeds at the time of their last update are spd_m and spd_n, and their positions are (x_m, y_m) and (x_n, y_n). Let t_{cur} represent the current time stamp, and let t represent the trigger time we want to obtain. Thus, we only need to solve a one-variable linear inequality to obtain t.

$$s - (t - t_{cur}) \cdot (spd_m + spd_n) - 2 \cdot r \leq \epsilon$$

where, $s = \sqrt{(x_m - x_n)^2 + (y_m - y_n)^2}$.

3.4.3.2 Efficient indexing of trigger times

We use two-level heap structures to index the trigger time, which is similar to [134]. Both the high-level and low-level heaps are min-heaps. For every object O_m, each of his friend pairs $\langle O_m, O_n \rangle$ corresponds to a trigger time $\omega(O_m, O_n)$, which is stored in a local min-heap $\mathcal{H}_m$ (the low-level heap), and the only pair which is pushed into the high-level heap $\mathcal{H}$, is the pair on the top of $\mathcal{H}_m$. By using this two-level heap indexing, the time complexity is efficiently decreased to $O(\log N + m \log m)$, which is much smaller than the one-level heap indexing method whose time complexity is $O(m \cdot \log \frac{N \cdot m}{2})$.

3.5 Experimental Study

We carry out experiments to compare the communication cost of our proposed algorithms.

3.5.1 Experimental Setting

The default value and range of each parameter used in our experiments can be found from Table 3.2. We generate moving objects by using the framework of network-based moving objects [17], on different road networks (one is Oldenburg* road network, and the other one is a part of New York city (NY)† road network, called pNY for short). In total, we generate N = 100,000 during 100 time stamps. We normalize the spatial domain size of the road networks to $[0, 1000]^2$.

Table 3.2: Parameter values.

Parameter	Meaning	Default	Range
N	Number of users	100,000	500 - 100,000
m	Number of friends per user	10	5 - 100
ϵ	Proximity threshold	10	1 - 100
V_{limit}	Maximum speed	42.8032	2.0 - 200
λ	Radius of mobile region	7.395	0.01 - 200
α	Tuning parameter	2	1 - 16
β	Tuning parameter	20/λ	>0 - 60

3.5.2 Experimental Study of FRMD

We carry out experiments to study the behavior of the FRMD cost model, as shown in Fig. 3.4(a) and (b), where m = 30, ϵ = 10, and ΔT = 1. Figure 3.4(a) and (b) depict the decomposition of the communication cost as a function of different fixed radius λ. Observe that the probing cost is high when λ is large and the update cost is high when λ is small. Without doubt, the notification cost is independent of λ. Meanwhile, we find there is an optimal λ which can minimize the total communication cost. This result coincides with our analysis in Section 3.2.4.

*http://www.cs.fsu.edu/%7Elifeifei/SpatialDataset.htm
†http://www.dis.uniroma1.it/challenge9/download.shtml

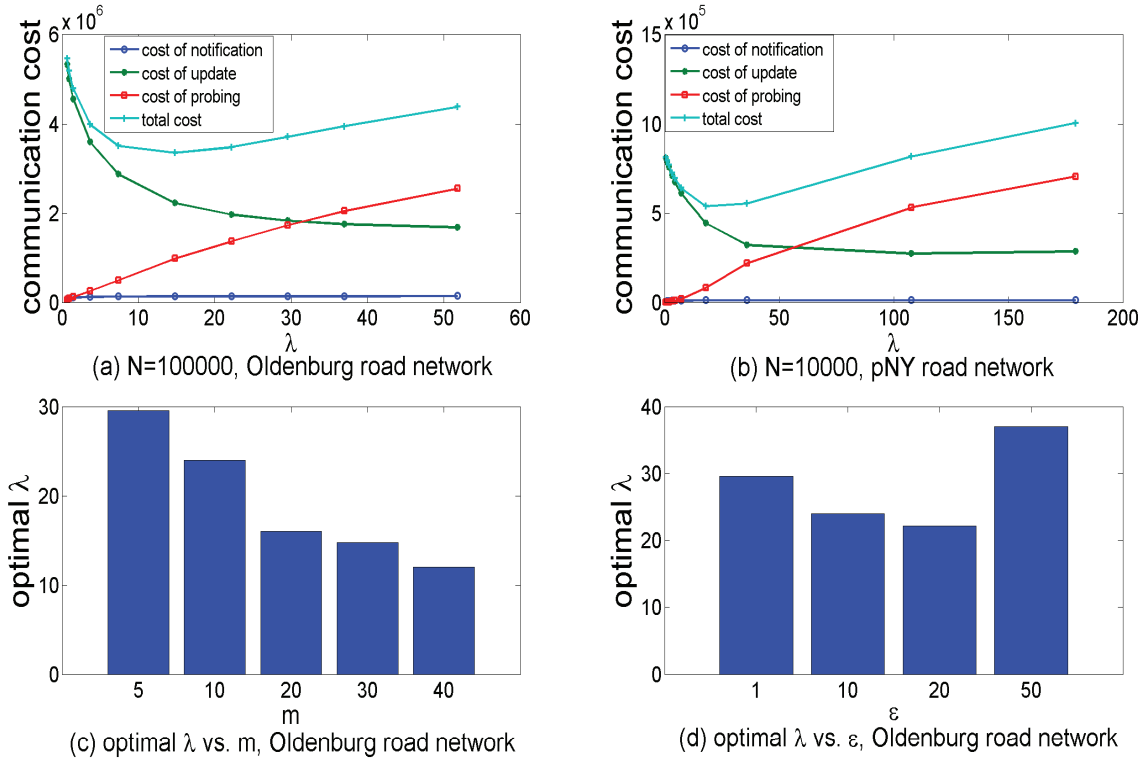


Figure 3.4: FRMD results.

Next, for each case, we conduct experiments to determine the optimal λ at which the communication cost can be minimized. We set N = 100,000. Figure 3.4(c) depicts the optimal λ as a function of the average number of friends m per user while $\epsilon = 10$, and Fig. 3.4(d) shows the optimal λ w.r.t. various ϵ while $m = 10$. Observe that the optimal λ depends on the value of m and ϵ that we select.

3.5.3 Self-Adjustment Experimental Study

We study the performance of our self-adjustment methods.

3.5.3.1 Sensitivity experiment

We first study the sensitivity of our FRMD, RMD_{RN}, CMD_{RN}, and RRMD methods. We compare the results of these methods w.r.t. (with respect to) parameter λ on both the Oldenburg and pNY road networks.

Figure 3.5 depicts the communication cost of these methods w.r.t. the initial radius λ of each mobile region. In the scenario of Fig. 3.5(a), the values of parameters such as N,

m, ϵ, V_{limit}, α and β are set as the default values in Table 3.2. In the scenario of Fig. 3.5(b), the number of users N on the pNY road network equals 10000, and the other parameters remain the same as that in Fig. 3.5(a). We observe that the shape of the curve representing the FRMD method coincides with the analysis in Section 3.2.4 and the RRMD method substantially reduces the communication cost of FRMD method to a greater extent than the RMD_{RN} and CMD_{RN} methods. Observe that the RMD_{RN} and CMD_{RN} methods oscillate wildly while RRMD method oscillates much more slightly. When the initial λ is large, the RRMD method can greatly reduce the communication cost but the RMD_{RN} and CMD_{RN} seem to be inefficient. In addition, our RRMD method reduces the communication cost of FRMD more greatly compared to the RMD_{RN} or CMD_{RN} method for all values of λ.

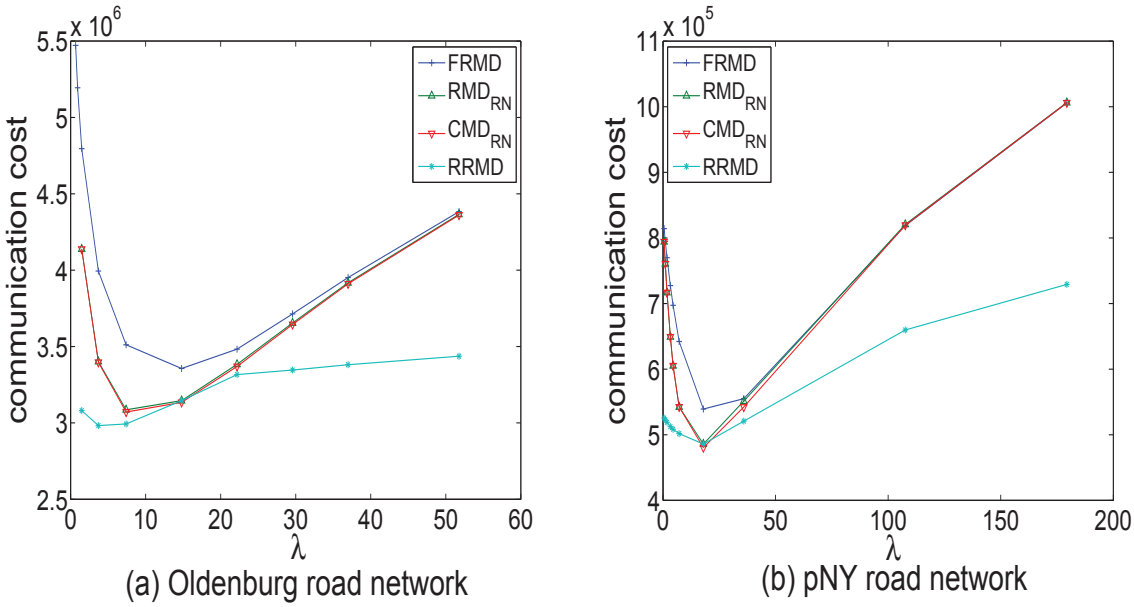


Figure 3.5: Communication vs. λ.

3.5.3.2 Scalability experiment

We subsequently compare our RRMD method to other competitors (RMD_{RN}/CMD_{RN}) with respect to various parameters such as N, V_{limit}, ϵ and m, on both the Oldenburg road network and pNY road network. Results demonstrate that our RRMD method is robust and scalable for these different parameters.

In the scenarios of Fig. 3.6, 3.7, 3.8, 3.9, the value of each parameter N, m, ϵ, α, β, λ

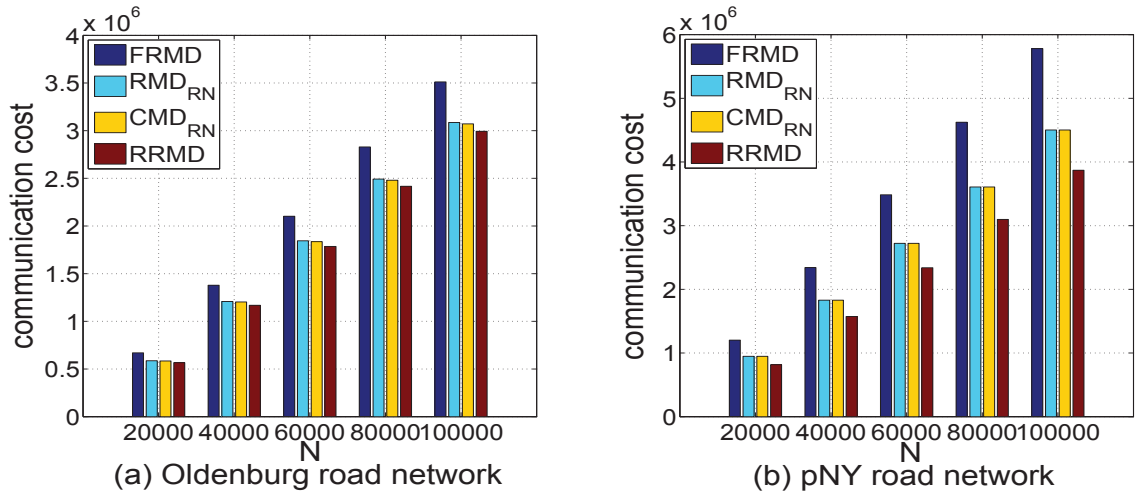


Figure 3.6: Communication cost vs. N.

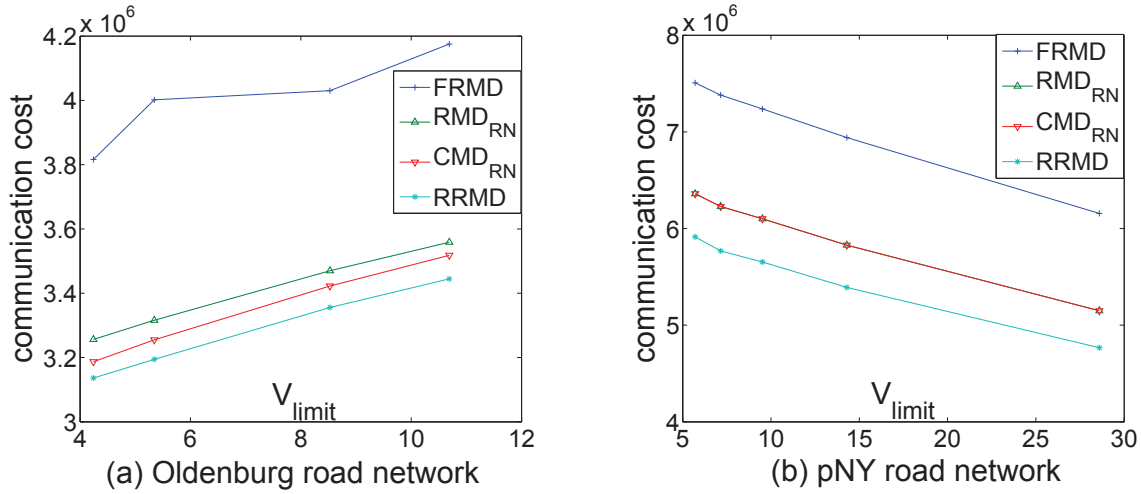


Figure 3.7: Communication cost vs. V_{limit}.

is set as the default value given in Table 3.2. Figure 3.6 depicts the communication cost induced by these methods with respect to N, which denotes the number of objects, and our RRMD method achieves the lowest cost of all the four methods. Figure 3.7 plots the communication cost induced by these methods as a function of the speed limit V_{limit} and observe that the RRMD method induces lower communication cost compared to other methods and scales linearly to V_{limit}. Figure 3.8 shows the communication cost incurred by these methods as a function of the threshold ϵ and we find the cost of our RRMD method is also the lowest of all the four methods. Figure 3.9 plots the communication cost incurred by these methods as a function of m, which denotes average number of friends for each user. It is obvious that the cost of our RRMD method also incurs the lowest

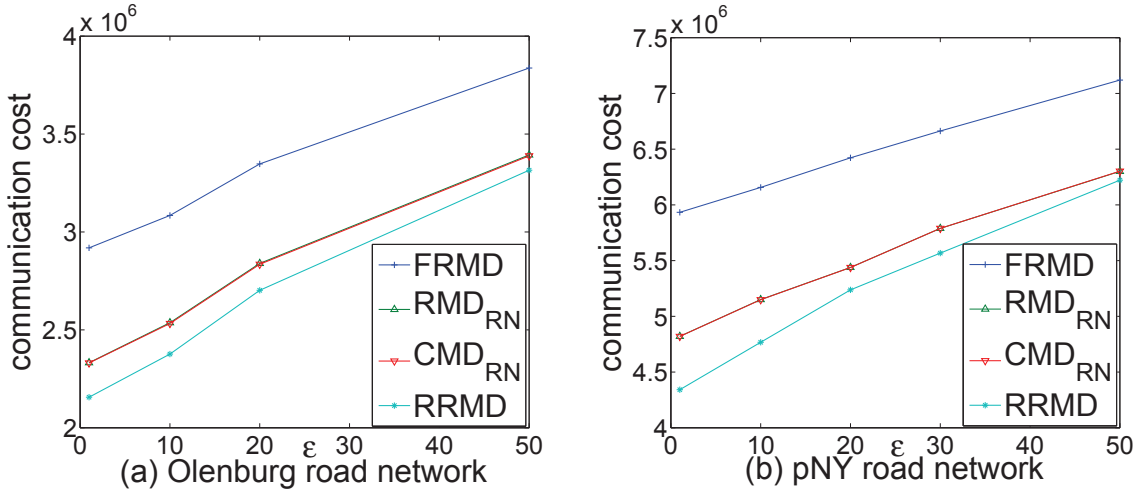


Figure 3.8: Communication cost vs. ϵ.

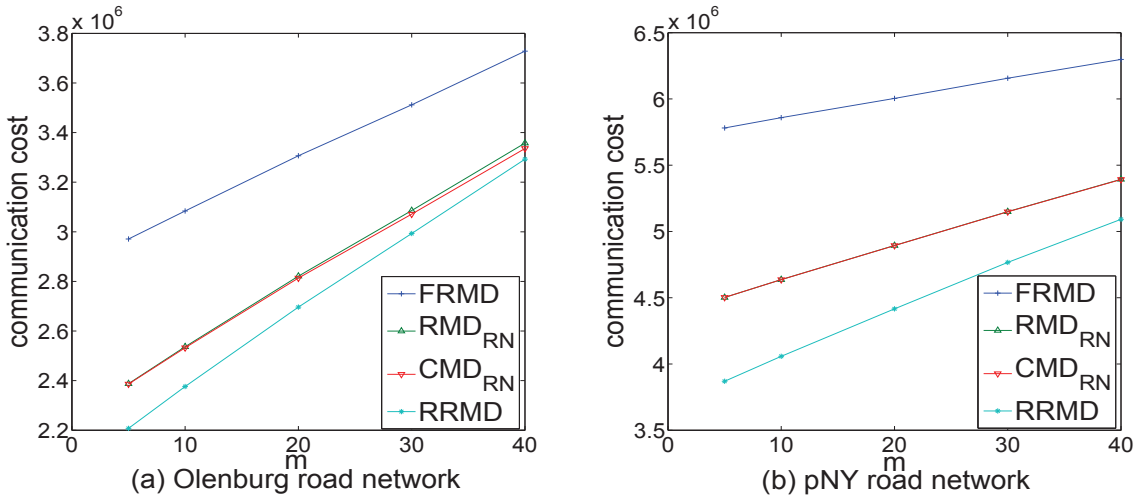


Figure 3.9: Communication cost vs. m.

communication cost among all the four methods and it scales approximately linearly to m.

3.5.3.3 Comparison with related work (CPMRN [68])

We also compare our results with the CPMRN [68]. As shown in Fig. 3.10, CPMRN incurs much more communication cost than our RRMD method. This is because in CPMRN, the server has to send messages to each client c to transfer information to it, even after the server builds the sets $P(c)$, $S(c)$, or after the server builds a proximity region and separation region for it. In addition, a client has two kinds of update messages, proximity alert and separation alert. Thus the update of a client is much more frequent than that in

our method.

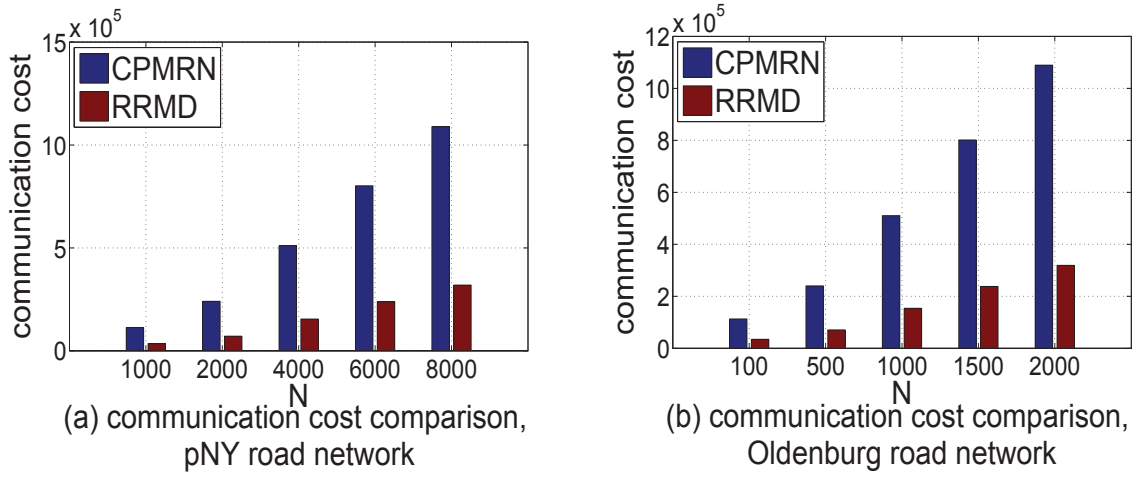


Figure 3.10: Comparison with CPMRN.

3.5.4 Experiments of Server-Side Computational Cost Optimization

3.5.4.1 Junction-to-junction network distance computation

We conduct experiments to compare the two methods described in Section 3.4.2 in terms of communication cost, server time and memory. From Fig. 3.11, we find that the server time of using Method 2 is much more than that of using Method 1. This is because this server time does not include the precomputation time in Method 1. The difference between the communication costs of Method 1 and Method 2 is small. Table 3.3 shows that Method 2 requires less memory than Method 1. If the road network is quite large, the memory used by Method 1 is much larger than that of Method 2 (see Table 3.3) and we can choose Method 2.

Table 3.3: Memory comparison.

road network	**memory of method 1 (K)**	**memory of method 2 (K)**
Oldenburg	3,507,252	116,820
pNY	25,456	6,176

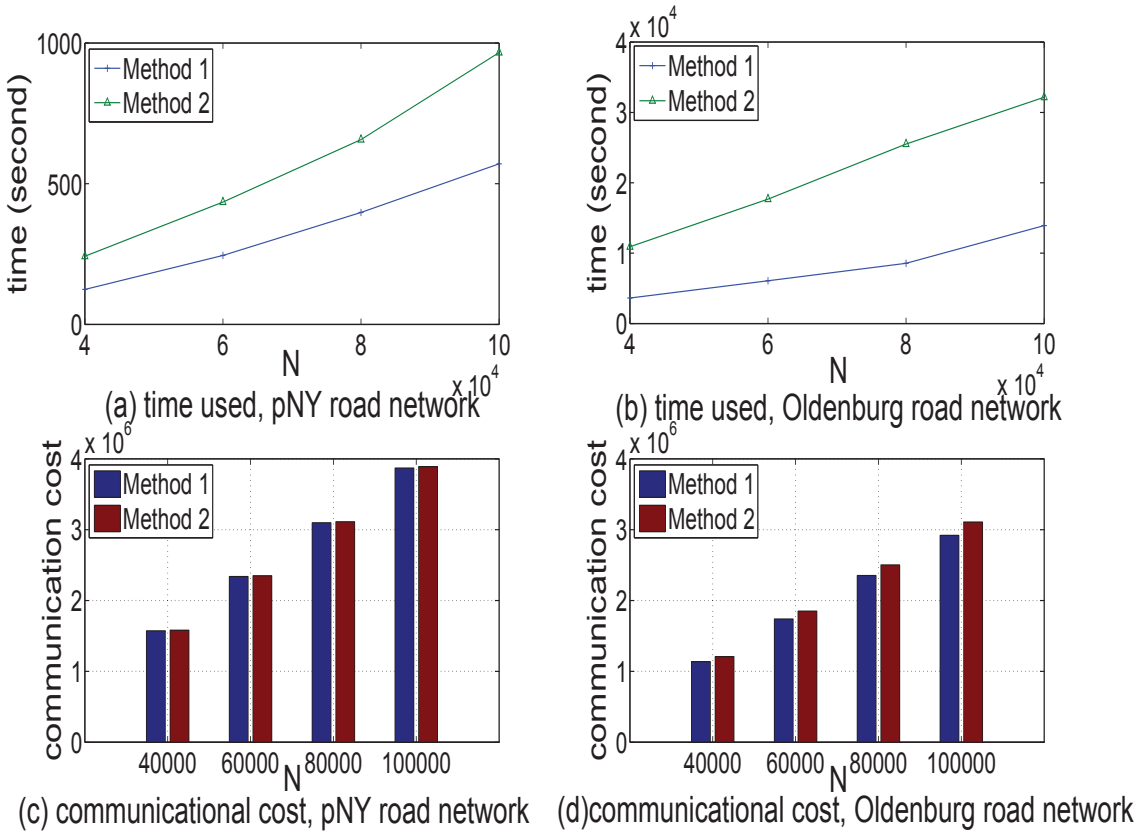


Figure 3.11: Time and communication cost comparison.

3.5.4.2 Experiments of trigger time technique

We conduct experiments on the pNY road network and the value of each parameter N, ϵ, α, β, λ is set as the default value in Table 3.2, with respect to different m. From Table 3.4, we observe that our trigger time technique (see Section 3.4.3) greatly reduces the total server time.

Table 3.4: Total server time.

m	**server time without trigger time technique (seconds)**	**server time with trigger time technique (seconds)**
5	562	167
10	933	176
20	2087	267
30	4911	297
40	8998	336

3.5.5 Experiments on Real-World Moving Objects

The experimental results reported above are based on generated moving objects in real-world road networks. As the movements of real objects are different from the generated objects, now we discuss the experiments on real-world moving objects. We take the taxi (T-Drive) trajectory data set[‡] as an example.

The real taxi trajectory data does not have speed information or edge information but the generated objects data contains the edge and speed information at each time stamp. However, from the real taxi trajectory data, we can compute the average speed during each fine time interval, e.g., every 5 seconds. We can also compute the direction of each moving object, and obtain the very edge on which the object is running at current time stamp, which makes the difference between the real-world data and the generated data small enough. Thus the difference between the movements of real-world objects and the movements of generated objects plays a minor role in the performance of our algorithms. Hence the communication cost model is similar. This difference would affect the relative performance by only a little bit, which is acceptable. The overall performance is still very similar. Therefore, here we omit the details of the experimental results on the real-world moving objects data set.

3.6 Conclusions

Motivated by various proximity detection applications and multiplayer online games in road networks, we propose two types of methods based on the client-server architecture to address the query of proximity detection in road networks. In the first type of method, we define a fixed-radius mobile region for each moving client. Unless a client exits its mobile region, the client need not update its location. In addition, we state three lemmas which are used to reduce the probing messages sent by the server. The size of the mobile region affects the update cost and probing cost. Inspired by this, we design the second kind of method RRMD together with RMD_{RN} and CMD_{RN} methods, using a self-adjustment

[‡]http://research.microsoft.com/apps/pubs/?id=152883

policy to automatically tune the size of the radius of the mobile region so as to reduce the total communication cost. Experiments demonstrate that our self-adjustment method can substantially reduce the cost, and is robust and scalable with respect to various parameters. In addition, we also propose optimization methods to reduce the total computational cost at the server side.

Another similar query is convoy query which finds many applications for traffic jam analysis. Therefore, for future work, we can adapt the methods proposed in this chapter to the problem of convoy query.

Chapter 4

Points of Interest Recommendation From GPS Trajectories

This chapter solves the problem of points of interest recommendation. This chapter is arranged as follows. Section 4.1 proposes the problem setting and our main framework to recommend POIs for moving users based on their GPS trajectories. Section 4.2 studies the efficiency and accuracy as well as recall of our method. Experiments prove that our framework outdoes the state-of-the-art method. Finally, Section 4.3 concludes this chapter.

4.1 Points of Interest Recommendation Framework

We present the problem setting and our recommendation methods in detail in this section.

4.1.1 Problem Setting and Framework Overview

Given a set of moving clients U, as well as their historical GPS trajectories *Traj*, we aim to recommend interesting semantic locations (POIs) l, based on the trajectory patterns of the clients, by taking popularity, temporal and geographical influences into consideration.

We propose a framework, namely, PTG-Recommend, with a vigorous probabilistic foundation for POI recommendation, based on density-based clustering algorithms and the Bayes rule. The novelty of our approach is that we recommend POIs from users' GPS trajectories rather than from a set of POIs. The advantage of our approach is that

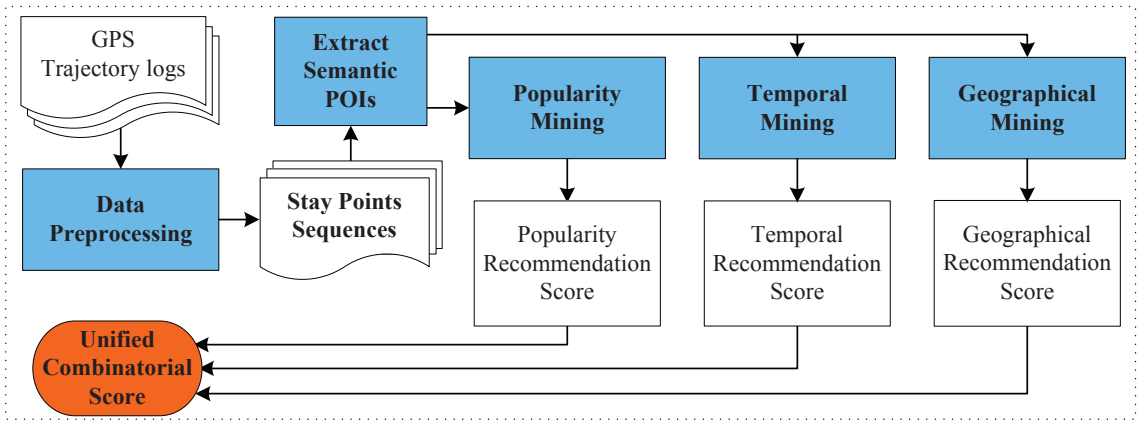


Figure 4.1: An overview of the unified framework PTG-Recommend.

we consider semantics when extracting POIs and exploit the popularity information as well as temporal and geographical effects together from GPS trajectories, leading to an improvement of POI recommendation accuracy.

Generally speaking, the unified PTG-Recommend framework comprises of the following six steps. (i) data preprocessing; (ii) extracting semantic POIs; (iii) popularity mining; (iv) temporal mining; (v) geographical mining; (vi) deriving a unified recommendation scoring function for extracted POIs. As shown in Fig. 4.1, this framework first transforms the raw GPS trajectories of each client into stay points sequences, and subsequently utilizes semantically enhanced clustering technique to extract semantic POIs. Then we get a popularity score by considering popularities of each POI; and derive a temporal recommendation score by considering temporal influence; and afterwards obtain a geographical recommendation score after geographic mining. Finally, we combine the three scores together for deriving a unified combinatorial score for each POI.

We give some definitions as follows prior to detailing the six steps of the unified framework.

Definition 9 ***GPS trajectory.*** *Each client's GPS trajectory is represented as a time-dependent sequence of triples in the form of (*latitude, longtitude, t*), which are collected from the GPS records.* ♡

Definition 10 ***Stay points.*** *A stay point is a geo-point where a client stays for a certain time period. A stay point consists of the latitude-longitude information of a geographical point in the form of a quadruple (latitude, longitude, t_{in}, t_{out}). We construct a mapping*

from each tuple (latitude, longitude) to a location point p. t_{in} *and* t_{out} *denote the checking-in and the checking-out time stamps, respectively.* ♡

Definition 11 ***Point of interest.*** *A point of interest refers to a location (region) which people are interested in, and which contains multiple stay points and the coordinates of the centroid of this region is the average of coordinates among all the stay points inside it. We obtain points of interest by clustering stay points. The detailed information is illustrated in Section 4.1.3.* ♡

Definition 12 ***Density-based*** ϵ ***neighborhood.*** *For a given point* p*, its density-based* ϵ *neighborhood is denoted by* $NB(p)$ *whose definition is as follows.*

$$\begin{cases} NB(p) & = \{o \in S | dist(p, o) \leq \epsilon\} \\ |NB(p)| & \geq MinPts \end{cases}$$

where S *denotes the stay points set,* o *denotes an arbitrary point in* S*,* ϵ *defines the density and represents the radius of the neighborhood circle centred at* p*, and MinPts represents the least number of stay points required in the neighborhood.* ♡

Definition 13 ***Density-joinable.*** *Set* A *is density-joinable to another set* B*, if there exists a point* o *belonging to both* A *and* B*.* ♡

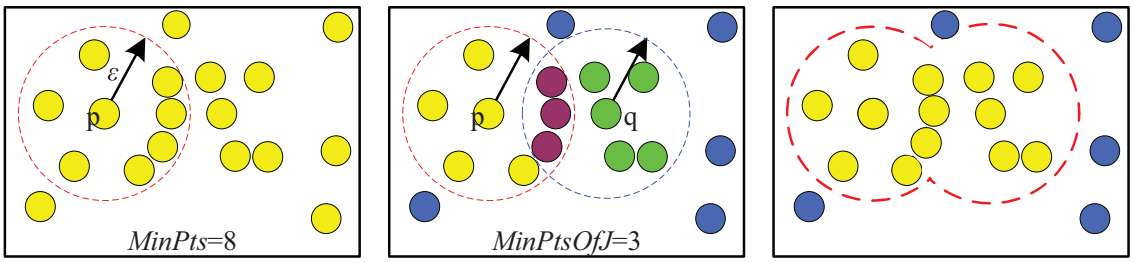
Definition 14 ***Density-threshold-based-joinable.*** *Given two sets* A *and* B*,* A *is density-threshold-based-joinable to* B*, if there are at least MinPtsOfJ stay points, that belong to both* A *and* B*.* ♡

Figure 4.2 illustrates the *density-based* ϵ *neighborhood* and *density-threshold-based-joinable* relation, where *MinPtsOfJ* = 3 and ϵ = 8.

4.1.2 Data Preprocessing

As depicted in Fig. 4.1, we first transform the raw GPS trajectory data of each user into stay points sequences. Raw trajectory data contains the latitude-longitude information of a location at a specified time stamp for a set of users. We record each item of the raw GPS trajectory data as a triple in the form of (*latitude*, *longitude*, *t*). Then each user's trajectory is represented as a sequence of triples.

In addition, we use a two-dimensional array to store the latitude-longitude tuple and construct a mapping from each tuple (*latitude*, *longitude*) to a location point *p*, which is



(a) Density - *NB(p)* (b) Density-Threshold-Based-Joinable (c) DTBJ-Cluster

Figure 4.2: Density-threshold-based-join (DTBJ) clustering. (a) *NB(p)* of a point p contains points inside a circle; (b) *NB(p)* is density-threshold-based-joinable to *NB(q)*; (c) the final cluster is inside the boundary.

represented by a numeric value (a key number). The advantage of using Maps is that we can acquire the latitude-longitude information from the key number with a time complexity of $O(\log n)$ and meanwhile we can obtain the key number of a location point p from the latitude-longitude information with the same time complexity $O(\log n)$, where n denotes the total number of the stay points. Therefore, each user has a sequence of stay location points.

For the purpose of obtaining qualified candidate stay points, we use two rules to filter out ineligible original stay points: (i) Stay points that are visited by target clients for less than a time threshold t_ϵ are filtered. (ii) Stay points that are visited by the clients for fewer than T times are filtered. After performing this preprocessing step, only those stay points visited by users for more than T times and more than a time threshold t_ϵ are selected into the candidate stay points sequences set.

4.1.3 Extracting Semantic POIs

We now present our methods of extracting the semantic POIs from candidate stay points sequences.

4.1.3.1 Extracting clusters

We use a three-layer model as shown in Fig. 4.3 to mine points of interest from GPS trajectories. Figure 4.3 describes the bottom-up process. First, the base layer is GPS trajectory. As described in Section 4.1.2, from users' GPS trajectories, we extract stay points from each of the GPS trajectories according to Definition 10. We use this def-

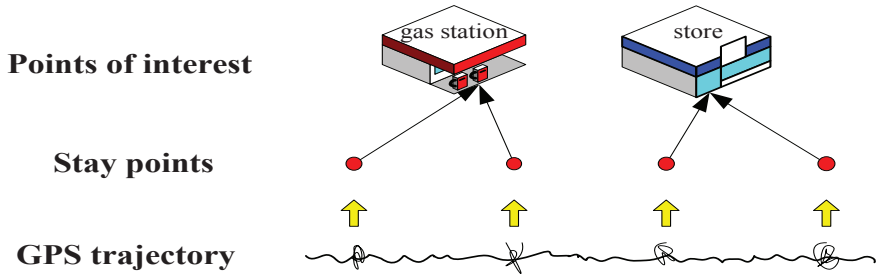


Figure 4.3: Three-layer model for extracting POIs.

inition to filter some invalid GPS points and select valid stay points. Second, after we obtain those stay points, according to Definition 11, we are ready to extract points of interest. Then based on the work ([144]) which presents a density-based clustering algorithm DJ-Cluster ([145]), we introduce a third parameter *MinPtsOfJ* and propose a new density-based clustering algorithm, namely, DTBJ-Cluster, and perform DTBJ-cluster on candidate stay points to obtain interesting locations (POIs).

DJ-Cluster. We briefly introduce the algorithm of DJ-Cluster. For every stay point p, DJ-Cluster calculates its density-based ϵ neighborhood. If it finds no neighborhood of p, p is marked as a noise; Otherwise, if there exists one of its neighbors belonging to an existing cluster c_i, the algorithm merges *NB(p)* with c_i; Otherwise, if none of its neighbors lies inside an existing cluster, the algorithm creates *NB(p)* as a new cluster.

DTBJ-Cluster. Our DTBJ-Cluster algorithm is devised on the basis of the aforementioned DJ-Cluster algorithm and the density-threshold-based-joinable property. In addition to the two parameters ϵ and *MinPts* which are already defined in DJ-Cluster, DTBJ-Cluster algorithm introduces a third parameter *MinPtsOfJ*, which represents the number of common points in the intersection area of *NB(p)* and its density-based cluster. The main idea of this algorithm is outlined in Lines 2 - 24 of Algorithm 1.

We calculate the density-based neighborhood *NB(p)* for an unprocessed stay point p with respect to the pre-determined parameters ϵ and *MinPts*. If *NB(p)* is empty which means no neighbors can be found for point p, then we make a new cluster for it. Otherwise, we find a set of density-joinable clusters C_{dj} for *NB(p)*. For each cluster $c_i \in C_{dj}$, if the number of common points between *NB(p)* and c_i is larger than or equal to *MinPtsOfJ*, then we merge *NB(p)* with cluster c_i. If for all the clusters $c_i \in C_{dj}$, the number of common

points between *NB(p)* and c_i is smaller than *MinPtsOfJ*, then a new cluster c is created based on *NB(p)*. Finally, if there are no density-joinable clusters of *NB(p)* existing, we also make a new cluster c on the basis of *NB(p)*.

4.1.3.2 Semantics-enhanced POIs extraction by SEM-DTBJ-Cluster

Given n stay points, after data preprocessing and DTBJ-Cluster, we extract some clusters. In order to get semantic POIs, many applications allow users to manually tag and comment these locations. However, man-made tagging causes users to spend time on tagging and thus should be avoided. Therefore, much existing work uses reverse geocoding technique to extract semantic information. Then following the idea of SEM-CLS (semantics-enhanced clustering) in [21], we apply two steps split and merge, to each cluster, and thus propose our **SEM-DTBJ-Cluster** (SEMantics-enhanced DTBJ-Cluster) algorithm (Algorithm 1).

In the split step (Lines 25 - 34 of Algorithm 1), points are sampled from each cluster. we then gain street addresses by reversely geocoding the sampled points using Google Map API. We subsequently gain the semantics using a yellow pages directory. In case the semantics of these sampled points inside one cluster are different, this cluster will be split in that it might contain multiple semantic POIs.

In the merge step (Lines 35 - 44 of Algorithm 1), the clusters will be merged if they refer to the same semantic POI. We use a semantic list vector $\vec{l_s}$ to contain all the semantic features of the cluster. We compare $\vec{l_s}$ of two clusters (Line 40) for determining whether they should be merged.

4.1.4 Applying Effect of Popularity

It is observed that in tourism, places of interest receive diverse popularity; in catering industry, restaurants obtain distinct popularity. Places of interest with high popularity may be more meaningful or valuable and restaurants with high popularity may produce high-quality food or more delicious food. From this point of view, POIs of high popularity should attract more tourists or customers. These are simple facts which demonstrate the

Algorithm 1: The SEM-DTBJ-Cluster Algorithm

Input: ϵ, *MinPts*, *MinPtsOfJ*, stay points set S, existing clusters set C_{dj}
Output: points of interest (a set of clusters) C

$C \leftarrow$ the set of clusters;
while $\exists p \in S$ *where* p *is unprocessed* **do**
  if *NB(p)* = $\emptyset$ **then**
    $p \leftarrow$ noise;
  end
  else if *NB(p) is density-joinable to a set of existing clusters* C_{dj} **then**
    flag $\leftarrow 0$;
    for i = *0 to* $|C_{dj}|$ **do**
      PtsOfJ $\leftarrow$ number of common points of c_i and *NB(p)*;
      if *PtsOfJ* $\geq$ *MinPtsOfJ* **then**
        $c_i \leftarrow c_i \cup$ *NB(p)*;
        flag $\leftarrow 1$;
      end
    end
    if *not flag* **then**
      $c \leftarrow$ *NB(p)*;
      C.push_back(c);
    end
  end
  else
    $c \leftarrow$ *NB(p)*;
    C.push_back(c);
  end
end
for i = *0 to* $|C|$ **do**
  Sample n stay points in c_i;
  for *each sample point* sp **do**
    Reverse geocode sp;
    Obtain semantics of sp;
  end
  if c_i *contains* n_s *different semantics* **then**
    Split c_i into n_s clusters according to semantics of each stay point;
  end
end
for i = *0 to* $|C|$ **do**
  for j = *0 to* $|C|$ **do**
    if $c_j = c_i$ **then**
      continue;
    end
    if $\vec{l_s}$ *of* $c_i = \vec{l_s}$ *of* c_j **then**
      $c_i \leftarrow c_i \cup c_j$;
    end
  end
end

existence of popularity effect over different POIs.

Let t_{cur} represent the current date, l stand for a semantic POI, and $x_{up}^{(t)}$ denote whether a user u visited stay point p at date t. We have

$$x_{up}^{(t)} = \begin{cases} 1, & u \text{ visited } p \text{ at } t, \\ 0, & u \text{ did not visit } p \text{ at } t. \end{cases}$$

Thus, let L denote the extracted POI set and N denote the number of semantic POIs in L. Then from users' trajectory history, we record a cumulative score to evaluate the popularity for each POI l over all users by the following scoring function.

$$\hat{c_{t,l}^{(p)}} = popScore_l^{(t)} = \frac{\sum\limits_{u \in U} \sum\limits_{0 \le t \le t_{cur} \&\& p \in l} x_{up}^{(t)}}{\sum\limits_{u \in U} \sum\limits_{0 \le i \le N} \sum\limits_{0 \le t \le t_{cur} \&\& p \in l_i} x_{up}^{(t)}} \tag{4.1}$$

where l_i denotes the i-th POI in set L. Note that in Eq. 4.1, we use the check-in history of each POI across all the users instead of a certain user u. This is because we assume that those POIs visited by more people tend to be more valuable. Therefore, we can recommend those POIs which get high popularity scores to users.

4.1.5 Applying Temporal Effect

In temporal mining, we utilize the periodic temporal property and partition time into periodic time slots (by date).

4.1.5.1 Exploiting temporal influence

It can be observed that a user goes to several identical or at least similar places day by day. For example, a person goes to a park for doing exercise every morning, and afterwards travels to his work place in the daytime, and subsequently goes to a specific restaurant for dinner everyday. These are similar or identical behaviors that we can easily find from a person's regular behaviors between this date and another date. In light of these facts, we are ready to exploit temporal influence on users' check-in behaviors by making use of the extracted POIs.

Given a client, client-based CF firstly computes the similarities between this client and other clients. Analogous to the general idea of [137], based on a weighted combination of other clients' access logs at each POI, we use CF to produce a temporal recommendation score for this POI. The conspicuous difference between our work and [137] is that they focus on the temporal influence on the POI recommendation from POIs data; whereas our work takes into account the popularity, temporal, and geographical influence together from GPS data and propose a unified framework. Another difference lies in the different metrics used. We use 'days' to measure time slots whereas [137] adopts 'hours' to measure time slots. We use 'days' in that we focus on the effects of 'days' on users' regular behaviors and 'days' better reveals the daily regularity and similarity.

To be more specific, suppose $v \in U$ is a client, and $l \in L$ is a POI where L is the set containing extracted POIs. If v has accessed l before, then let $c_{v,l}$ = 1; otherwise, $c_{v,l}$ = 0. Thus, for a client u, the recommendation score of u visiting l can be calculated as follows, where $s_{u,v}$ denotes the similarity of client u and client v.

$$\widehat{c_{u,l}} = \frac{\sum_v s_{u,v} c_{v,l}}{\sum_v s_{u,v}}$$

$s_{u,v}$ can be calculated by various measures. Among all the measures, we adopt a widely used measure, namely, cosine similarity for implicit data. Eq. 4.2 defines the cosine similarity of u and v, where a binary accessing vector over the entire POI set L is used to represent each client.

$$s_{u,v} = \frac{\sum_l c_{u,l} c_{v,l}}{\sqrt{\sum_l c_{u,l}^2} \sqrt{\sum_l c_{v,l}^2}} \tag{4.2}$$

We split the total time period into multiple equal time intervals based on dates. We utilize a user-date-POI (UDP) cube to store the temporal accessing records. Each element $c_{v,t,l}$ of the UDP cube specifies whether a user v visits POI l on date t, where

$$c_{v,t,l} = \begin{cases} 1; & \text{if } v \text{ visits } l \text{ at date } t \\ 0; & \text{elsewise} \end{cases}$$

Thus, the recommendation score for user u to visit a POI l on date t is:

$$\widehat{c_{u,t,l}^{(t)}} = \frac{\sum_v s_{u,v}^{(t)} c_{v,t,l}}{\sum_v s_{u,v}^{(t)}}$$

where $s_{u,v}^{(t)}$ refers to the temporal behavior similarity of u and v.

Now, we present the process of computing $s_{u,v}^{(t)}$ in detail. We estimate the similarity of two users on the basis of their daily regular behaviors over all dates. It is straightforward that the similarity value will be high, if the clients access the same POIs at the same date with high frequency. Moreover, in this case, one client's access history will significantly influence the recommendation score for the other client. Thus, the similarity of u and v can be calculated by extending the cosine similarity as expressed in Eq. 4.3.

$$s_{u,v}^{(t)} = \frac{\sum_{t=1}^{T}\sum_{l=1}^{L} c_{u,t,l} \cdot c_{v,t,l}}{\sqrt{\sum_{t=1}^{T}\sum_{l=1}^{L} c_{u,t,l}^2}\sqrt{\sum_{t=1}^{T}\sum_{l=1}^{L} c_{v,t,l}^2}} \tag{4.3}$$

Suppose client u visits l_1 and l_2 at dates t_1 and t_2, while client v visits l_1 and l_2 at dates t_2 and t_1, respectively. By Eq. 4.2, the similarity between the two clients is 1 if irrespective of time. However, the similarity becomes 0 by Eq. 4.3 when we take time into consideration. This is because compared to the user-POI matrix, the UDP cube is much sparser. Note that in Eq. 4.3, we estimate the similarity by considering temporal effect based on the UDP cube.

In order to handle this problem stemmed from sparsity, let $c_{u,t}$ = $\{c_{u,t,1}$, $c_{u,t,2}$, ..., $c_{u,t,l}\}$ represent the accessing vector of client u on date t. For every client u, we compute the cosine similarity of every two accessing vectors c_{u,t_i} and c_{u,t_j} on dates t_i and t_j (see Eq. 4.4). Thereafter, let $\lambda_{t_i t_j}$ denote the similarity value between two dates t_i and t_j, which refers to the average of the similarities of t_i and t_j over all clients (Eq. 4.5).

$$s_{t_i t_j}^{(u)} = \frac{\sum_{l=1}^{L} c_{ut_i l} \cdot c_{ut_j l}}{\sqrt{\sum_{l=1}^{L} c_{ut_i l}^2}\sqrt{\sum_{l=1}^{L} c_{ut_j l}^2}} \tag{4.4}$$

$$\lambda_{t_i t_j} = \frac{1}{U} \cdot \sum_{u=1}^{U} s_{t_i t_j}^{(u)} \tag{4.5}$$

Figure 4.4 shows the cosine similarity between four different given dates (Date 27, Date

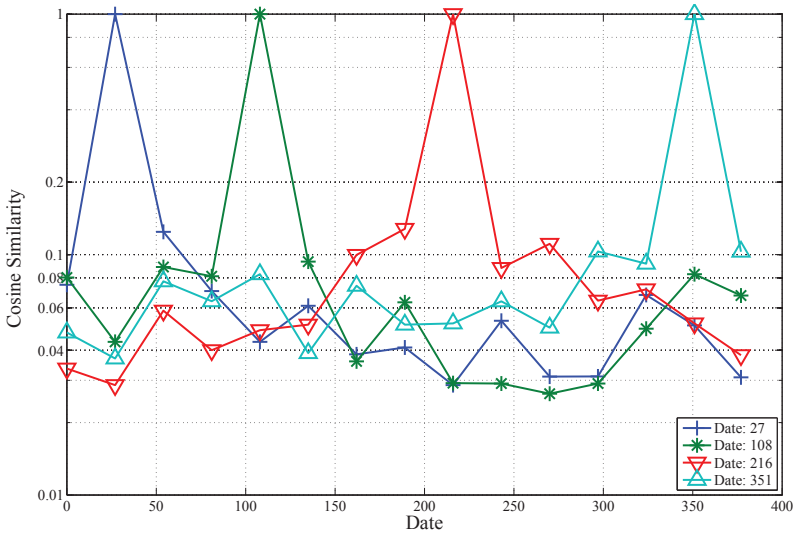


Figure 4.4: Mobile user behavior similarities between different dates.

108, Date 216, Date 351) on Geolife trajectory dataset*, which has a number of 182 mobile users and the whole time period is more than five years. We divide each year to 365 days. Each record in this dataset records which user visits which point at what time (hour:min:sec) on which day. (Section 4.2.1 gives the details of this dataset.) In this figure, we describe the date-date similarities for all users. Observe that the similarity curve for Date 27 depicts the accessing similarity between Date 27 and every other date during the whole period, analogously for the other three curves. We see that for a specific user, the similarity between two close dates, for instance, the similarity of Date 27 and Date 27 - 54 is much higher than that between Date 27 and another date far away. To sum up, the accessing activity on one date is more analogous to that on a few dates than other dates. The fact encourages us to devise a framework with temporal factor utilized for POI recommendations.

By Eq. 4.5, the UDP cube is smoothed by using the accessing similarity of different dates. Every accessing vector can be calculated by utilizing the accessing vectors on similar dates. Thus, the value of $c_{u,t,l}$ is updated as Eq. 4.6. Consequently, the similarity of two clients u and v enhanced by smoothing is computed in Eq. 4.7.

$$\tilde{c}_{u,t,l} = \sum_{t'=1}^{T} \frac{\lambda_{t,t'}}{\sum_{t''=1}^{T} \lambda_{t,t''}} c_{u,t',l} \tag{4.6}$$

*http://research.microsoft.com/en-us/downloads/b16d359d-d164-469e-9fd4-daa38f2b2e13/

$$\tilde{s}_{u,v}^{(t)} = \frac{\sum_{t=1}^{T}\sum_{l=1}^{L}\tilde{c}_{u,t,l}\cdot\tilde{c}_{v,t,l}}{\sqrt{\sum_{t=1}^{T}\sum_{l=1}^{L}\tilde{c}_{u,t,l}^{2}}\sqrt{\sum_{t=1}^{T}\sum_{l=1}^{L}\tilde{c}_{v,t,l}^{2}}} \tag{4.7}$$

The accessing behavior similarity between two dates additionally helps recommend POIs to user u on date t by introducing another enhancement. If user v visits POI l at date t', then the temporal scoring function, $\hat{c_{u,t,l}^{(t)}}$, that a user u visits the same POI l at a different date t is therefore updated as Eq. 4.8.

$$\hat{c_{u,t,l}^{(t)}} = \frac{\sum_{v}\tilde{s}_{u,v}^{(t)}\sum_{t'}\tilde{c}_{v,t',l}\cdot\lambda_{t,t'}}{\sum_{v}\tilde{s}_{u,v}^{(t)}} \tag{4.8}$$

4.1.5.2 A running example

Given: Let $T = \{1, 2\}$, $U = \{u, v\}$, and $L = \{1, 2\}$. User u checks in POI 1 at t=1; checks in POI 2 at t=2. Similarly, user v visits POI 1, 2 at t=1; visits POI 2 at t=2.

Goal: To derive the temporal recommendation score for user u to visit POI 1 at date t=2, that is, to compute $\hat{c_{u,2,1}^{(t)}}$.

Analysis: we can derive the following equation by expanding Eq. 4.8.

$$\hat{c_{u,2,1}^{(t)}} = \frac{\tilde{s}_{u,u}^{(t)}\cdot(\tilde{c}_{u,1,1}\cdot\lambda_{21}+\tilde{c}_{u,2,1}\cdot\lambda_{22}) + \tilde{s}_{u,v}^{(t)}\cdot(\tilde{c}_{v,1,1}\cdot\lambda_{21}+\tilde{c}_{v,2,1}\cdot\lambda_{22})}{\tilde{s}_{u,u}^{(t)}+\tilde{s}_{u,v}^{(t)}} \tag{4.9}$$

Therefore, to compute $\hat{c_{u,2,1}^{(t)}}$, we need to first compute $\tilde{s}_{u,u}^{(t)}$, $\tilde{s}_{u,v}^{(t)}$, $\tilde{c}_{u,1,1}$, $\tilde{c}_{u,2,1}$, $\tilde{c}_{v,1,1}$, $\tilde{c}_{v,2,1}$, λ_{21}, and λ_{22}.

First, based on the given conditions, we update values in the UDP cube.

$$\mathbf{c_{u,t,l}} = \begin{bmatrix} c_{u,1,1} & c_{u,1,2} \\ c_{u,2,1} & c_{u,2,2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{c_{v,t,l}} = \begin{bmatrix} c_{v,1,1} & c_{v,1,2} \\ c_{v,2,1} & c_{v,2,2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

After substituting corresponding $c_{u,t,l}, c_{v,t,l}$ into Eq. 4.4, we obtain the following:

$$\mathbf{s^{(u)}} = \begin{bmatrix} s_{11}^{(u)} & s_{12}^{(u)} \\ s_{21}^{(u)} & s_{22}^{(u)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{s^{(v)}} = \begin{bmatrix} s_{11}^{(v)} & s_{12}^{(v)} \\ s_{21}^{(v)} & s_{22}^{(v)} \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 1 \end{bmatrix}$$

Where, each element of the above two matrices represents $s_{ij}^{(u)}$, and $s_{ij}^{(v)}$, respectively. Ac-

cording to Eq. 4.5, we can compute $\lambda_{ij}, i, j \in \{1, 2\}$.

$$\lambda = \begin{bmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \end{bmatrix} = \begin{bmatrix} 1 & \frac{\sqrt{2}}{4} \\ \frac{\sqrt{2}}{4} & 1 \end{bmatrix} \approx \begin{bmatrix} 1 & 0.353553 \\ 0.353553 & 1 \end{bmatrix}$$

We substitute λ_{ij} into Eq. 4.6, to compute the new values of $c_{u,t,l}$ as follows.

$$\tilde{\mathbf{c}}_{\mathbf{u,t,l}} = \begin{bmatrix} \tilde{c}_{u,1,1} & \tilde{c}_{u,1,2} \\ \tilde{c}_{u,2,1} & \tilde{c}_{u,2,2} \end{bmatrix} = \begin{bmatrix} 0.738796 & 0.261204 \\ 0.261204 & 0.738796 \end{bmatrix}$$

$$\tilde{\mathbf{c}}_{\mathbf{v,t,l}} = \begin{bmatrix} \tilde{c}_{v,1,1} & \tilde{c}_{v,1,2} \\ \tilde{c}_{v,2,1} & \tilde{c}_{v,2,2} \end{bmatrix} = \begin{bmatrix} 0.738796 & 1 \\ 0.261204 & 1 \end{bmatrix}$$

By using Eq. 4.7, we obtain:

$$\tilde{s}^{(t)}_{u,u} = \frac{\sum_{t=1}^{2}\sum_{l=1}^{2} \tilde{c}_{u,t,l} \cdot \tilde{c}_{u,t,l}}{\sqrt{\sum_{t=1}^{2}\sum_{l=1}^{2} \tilde{c}^2_{u,t,l}}\sqrt{\sum_{t=1}^{2}\sum_{l=1}^{2} \tilde{c}^2_{u,t,l}}} = \frac{\sum_{t=1}^{2}\sum_{l=1}^{2} \tilde{c}^2_{u,t,l}}{\sum_{t=1}^{2}\sum_{l=1}^{2} \tilde{c}^2_{u,t,l}} = 1$$

$$\begin{aligned} \tilde{s}^{(t)}_{u,v} &= \frac{\sum_{t=1}^{2}\sum_{l=1}^{2} \tilde{c}_{u,t,l} \cdot \tilde{c}_{v,t,l}}{\sqrt{\sum_{t=1}^{2}\sum_{l=1}^{2} \tilde{c}^2_{u,t,l}}\sqrt{\sum_{t=1}^{2}\sum_{l=1}^{2} \tilde{c}^2_{v,t,l}}} \\ &= \frac{\tilde{c}_{u,1,1}\tilde{c}_{v,1,1} + \tilde{c}_{u,1,2}\tilde{c}_{v,1,2} + \tilde{c}_{u,2,1}\tilde{c}_{v,2,1} + \tilde{c}_{u,2,2}\tilde{c}_{v,2,2}}{\sqrt{\tilde{c}^2_{u,1,1} + \tilde{c}^2_{u,1,2} + \tilde{c}^2_{u,2,1} + \tilde{c}^2_{u,2,2}}\sqrt{\tilde{c}^2_{v,1,1} + \tilde{c}^2_{v,1,2} + \tilde{c}^2_{v,2,1} + \tilde{c}^2_{v,2,2}}} \\ &= 0.900832 \end{aligned}$$

Up to now, all the values in Eq. 4.9 are known. Hence, we substitute these values into Eq. 4.9 to obtain $\hat{c^{(t)}_{u,2,1}} = 0.522408$.

4.1.6 Applying Geographical Effect

When dealing with POI recommendation problems, the geographical influence is bound to be considered. As far as we know, many events that occur conform to a normal distribution. [25] utilized Gaussian distribution to model users' check-in behavior and denoted the normalized probability of a check-in stay point which belongs to the clustered regions. In our work, we utilize normal distribution in a different way.

It is obvious that clients tend to visit nearby places rather than a place further away.

Therefore, we adopt a normal distribution model to characterize a client's willingness to access a POI $dist$ kilometers away. Suppose $\mu = 0$, $\sigma^2 = 1$ (σ^2 may be also equal to other values) are the parameters of the normal distribution, since a normal distribution model can demonstrate the property that the nearer a place is, the greater the client's willingness to visit this place. Let $x = dist(l_i, l_j)$, then $x \propto N(0, \sigma^2)$. The density function of the normal distribution is defined as follows.

$$will(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp -\frac{x^2}{2\sigma^2}$$

As depicted in Fig. 4.5, when $x \geq 0$, the value of this function decreases monotonically

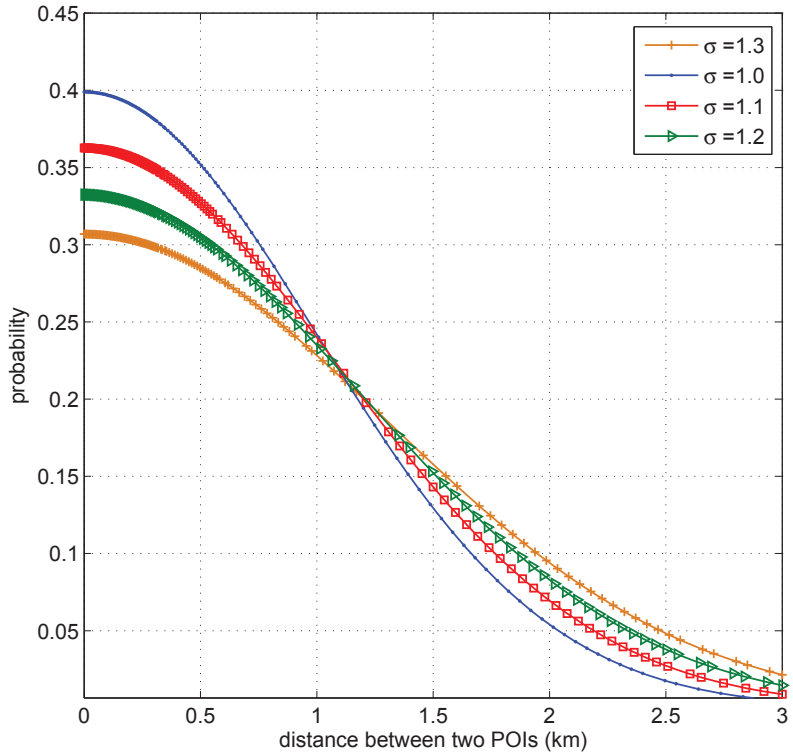


Figure 4.5: Normal distribution of probabilities.

and accordingly with respect to different σ. Here, σ can be regarded as a second factor that affects people's willingness in addition to the distance factor. For instance, a user might sometimes take into consideration both the distance and the attraction or other features of a POI. However, the willingness of a user complies with the normal distribution. Suppose currently a client is at POI l_i. The next POI to visit is l_j, which is $dist(l_i, l_j)$ away from l_i. Then the probability of the client visiting l_j is in proportion to the client's willingness

to visit a POI which is $dist(l_i, l_j)$ away from l_i. The conditional probability is calculated by using the following equation (Eq. 4.10).

$$prob(l_j|l_i) = \frac{will(dist(l_i, l_j))}{\sum_{l_k \in L} will(dist(l_i, l_k))} \tag{4.10}$$

The equation above shows the conditional probability will decrease if the distance between two POIs increases. This indicates that clients are less probable to access far-away POIs. Based on Bayes rule, the score for recommending a POI l to a given client u on the basis of his/her POIs trajectory L_u is obtained as Eq. 4.11.

$$\begin{aligned} \hat{c^{(g)}_{u,t,l}} &= \sum_{l' \in L_u} prob(l|l') \\ &= \sum_{l' \in L_u} \frac{will(dist(l', l))}{\sum_{l_k \in L} will(dist(l', l_k))} \end{aligned} \tag{4.11}$$

4.1.7 Unified Recommendation Score for POIs

Finally, we derive a unified scoring function for POI l, by applying a linear weighting on the three scoring functions (Eq. 4.1, 4.8, and 4.11). Since the three scoring functions are based on different measures and differ from each other, a min-max normalization is used to normalize the three scoring functions before we combine them.

$$\overline{c^{(p)}_{u,t,l}} = \frac{\hat{c^{(p)}_{t,l}} - min_{l'}(\hat{c^{(p)}_{t,l'}})}{max_{l'}(\hat{c^{(p)}_{t,l'}}) - min_{l'}(\hat{c^{(p)}_{t,l'}})} \tag{4.12}$$

$$\overline{c^{(t)}_{u,t,l}} = \frac{\hat{c^{(t)}_{u,t,l}} - min_{l'}(\hat{c^{(t)}_{u,t,l'}})}{max_{l'}(\hat{c^{(t)}_{u,t,l'}}) - min_{l'}(\hat{c^{(t)}_{u,t,l'}})} \tag{4.13}$$

$$\overline{c^{(g)}_{u,t,l}} = \frac{\hat{c^{(g)}_{u,t,l}} - min_{l'}(\hat{c^{(g)}_{u,t,l'}})}{max_{l'}(\hat{c^{(g)}_{u,t,l'}}) - min_{l'}(\hat{c^{(g)}_{u,t,l'}})} \tag{4.14}$$

$min_{l'}(\cdot)$ and $max_{l'}(\cdot)$ in the above equations denote the minimum visiting score and maximum visiting score for client u on date t over all POIs.

Subsequently, the unified recommendation scoring function for a client u to visit POI l on date t is given as follows, where α and β are tuning parameters and $0 \leq \alpha \leq 1$,

Figure 4.6: Distribution of stay points in Geolife trajectory dataset.

$0 \leq \beta \leq 1$.

$$Score_{u,t,l} = \alpha * \overline{c^{(p)}_{u,t,l}} + \beta * \overline{c^{(t)}_{u,t,l}} + (1 - \alpha - \beta) * \overline{c^{(g)}_{u,t,l}} \tag{4.15}$$

According to Eq. 4.15, we can compute the unified recommendation scores for all POIs. Finally we return the POIs with the highest scores to the client.

4.2 Experimental Study

Several groups of experiments are conducted to evaluate our PTG-Recommend method. We now report the experimental results. We firstly present two trajectory datasets that we use in the experiments, the training data and testing data distribution of each dataset. We then study the changes of the number of stay points along with T (times of visiting) and t_ϵ (stay duration) in the preprocessing step. We subsequently study the performance of our SEM-DTBJ-Cluster algorithm for extracting semantic POIs. We Finally evaluate the precision and recall of our POI recommendation methods: the method equipped with popularity influence, method equipped with temporal influence, method equipped with geographical influence, and the final unified method.

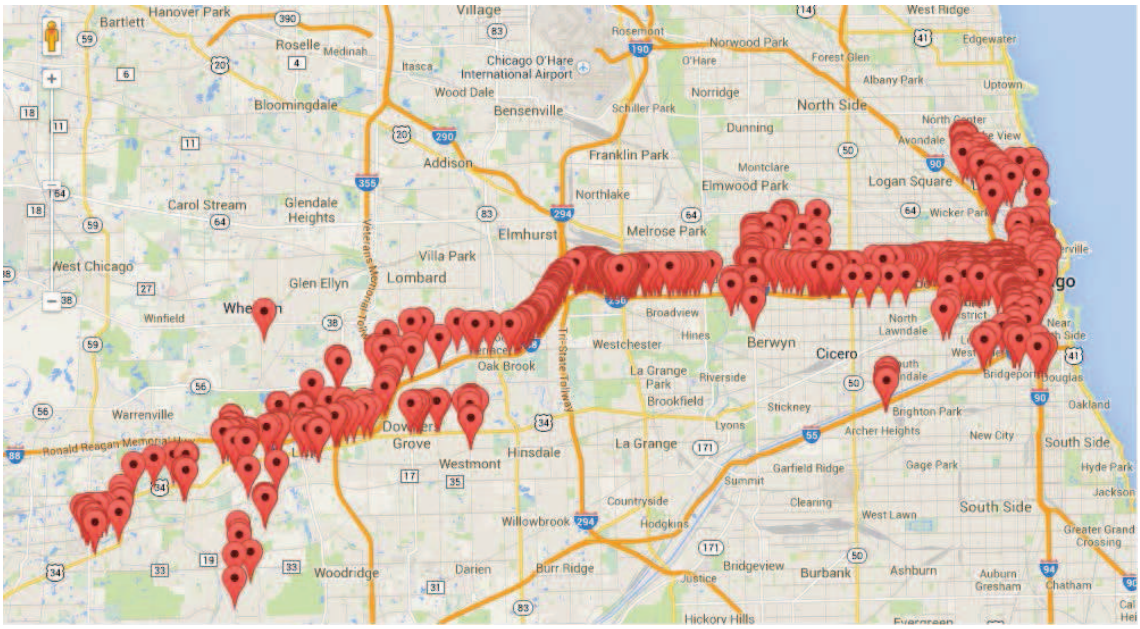


Figure 4.7: Distribution of stay points in Illinois trajectory dataset.

4.2.1 Experimental Setup

Datasets. The datasets we used are two real-world trajectory datasets: Geolife Trajectories 1.3[†] and Illinois real trajectory dataset[‡]. The distribution of the stay points of these two datasets are illustrated in Fig. 4.6 and 4.7. Table 4.1 lists the statistics of these two datasets.

Table 4.1: Information about the Geolife trajectory dataset and Illinois trajectory dataset (after pre-processing).

Dataset	#Check-ins	#Users	#Trajectories	#Nodes
Geolife	24,876,978	182	17,621	22,276,521
Illinois	357,786	2	124	313,239

Geolife trajectory dataset was obtained by 182 users at Microsoft Research Asia for more than five years (from April 2007 to August 2012). Different GPS loggers and GPS-phones were used to record the trajectories in a dense representation. In addition, the trajectories have various sampling rates. 17,621 trajectories with a total duration of 48,000+ hours and a total distance of roughly 1.2 million *km* are contained in this dataset. We use a sequence of points to represent each GPS trajectory. These points contains the infor-

[†]http://research.microsoft.com/en-us/downloads/b16d359d-d164-469e-9fd4-daa38f2b2e13/
[‡]http://www.cs.uic.edu/%7Eboxu/mp2p/gps%5Fdata.html

mation of latitude, longitude, altitude and the time stamps. The Geolife dataset originally contains 24,876,978 geographical points. We select the points collected by the first 130 users as training data which contains 18,516,628 points in total, and the points collected by the remaining 32 users as testing data, whose distribution is shown in Fig. 4.8(a) and (b).

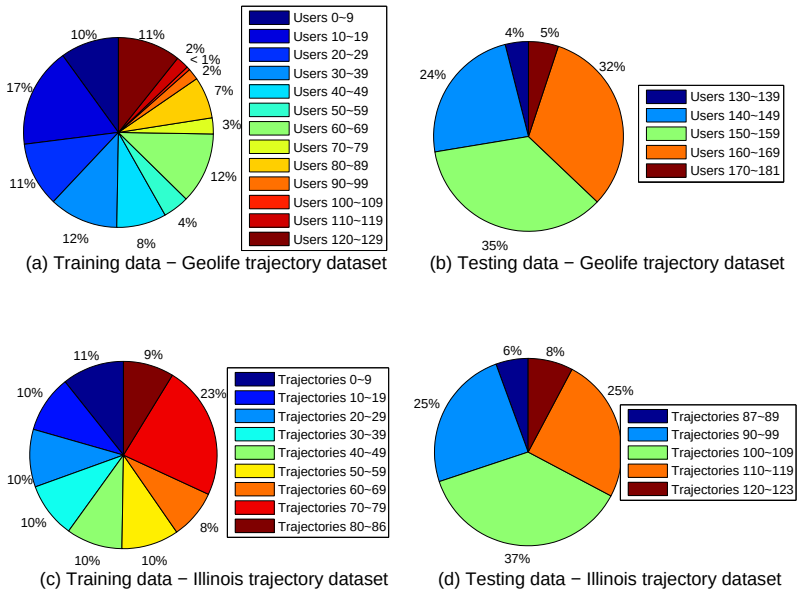
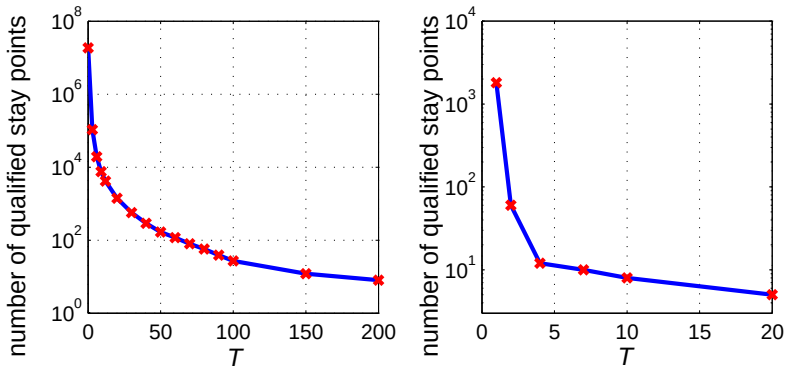


Figure 4.8: Training data and testing data distribution over users.

The Illinois trajectory dataset was collected in 2006 by two members of Databases and Mobile Computing Laboratory at University of Illinois in Chicago. Each trajectory records a continuous journey around the Dupage county and the Cook county of Illinois. Each record (sampled every second) of a trajectory contains latitude, longitude, time stamps, and ($x\_projection$, $y\_projection$) which are the coordinates projected from (latitude, longitude) by the projection of NAD 1983 HARN StatePlane Illinois East included in ESRI ArcView 3.1. This dataset originally contains 357,786 points before the preprocessing step. We select the data of the first 87 trajectories as training data and the remaining trajectories as testing data, whose distribution is shown in Fig. 4.8(c) and (d).

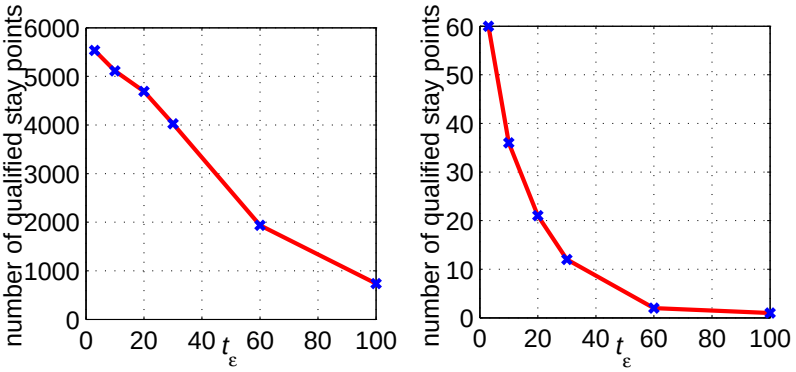
4.2.2 Preprocessing

In the preprocessing step, we evaluate the number of qualified stay points as a function of T (times of visiting) and t_ϵ (stay duration), as shown in Fig. 4.9 and 4.10. In the scenario of



(a) Geolife trajectory data (b) Illinois trajectory data

Figure 4.9: T versus number of stay points. t_ϵ = 3.



(a) Geolife trajectory data (b) Illinois trajectory data

Figure 4.10: t_ϵ versus number of stay points. (a) T=10; (b) T=2.

Fig. 4.9, stay duration t_ϵ is set to 3 seconds. In the scenario of Fig. 4.10, visiting times T is set as 10 for Geolife trajectory dataset, and 2 for Illinois trajectory dataset. From Fig. 4.9 and 4.10, we see that much more points have been filtered out with the increase of T or t_ϵ. For example, for Geolife trajectory dataset, when t_ϵ=3 seconds, and T=3 times, after filtering out unqualified points, 107,335 stay points have left, and when t_ϵ=3 seconds and T=10 times, after we have filtered out ineligible geographical points in data preprocessing step, 5,113 points were left. For Illinois trajectory dataset, when t_ϵ=3 seconds, T=1, 1,811 stay points are qualified; when T=2, t_ϵ=3 seconds, 60 stay points are qualified.

4.2.3 Comparing DTBJ-Cluster with DJ-Cluster

We subsequently compare our DTBJ-Cluster algorithm with the existing DJ-Cluster algorithm. The results on Geolife trajectory dataset and Illinois trajectory dataset are reported in Table 4.2 and Table 4.3. Table 4.2 plots the number of clusters on Geolife dataset as a

Table 4.2: Number of clusters with respect to different clustering algorithms and various parameters such as ϵ, *MinPts*, and *MinPtsOfJ*, on Geolife trajectory dataset. T = 6 times, t_ϵ = 6 seconds.

		DJ-Cluster						**DTBJ-Cluster**					
ϵ(km)		20			30			20			30		
MinPts		3	4	5	3	4	5	3	4	5	3	4	5
# **Clusters (without semantics)**	**MinPtsOfJ=2**	356	309	272	271	235	209	394	336	297	294	253	226
	MinPtsOfJ=4	356	309	272	271	235	209	432	369	324	319	276	240
	MinPtsOfJ=6	356	309	272	271	235	209	475	414	371	330	286	250
	MinPtsOfJ=8	356	309	272	271	235	209	510	452	410	344	300	265

Table 4.3: Number of clusters with respect to different clustering algorithms and various parameters such as ϵ, *MinPts*, and *MinPtsOfJ*, on Illinois trajectory dataset. T=1 time, t_ϵ=1 second).

		DJ-Cluster						**DTBJ-Cluster**					
ϵ (metres)		20			30			20			30		
MinPts		3	4	5	3	4	5	3	4	5	3	4	5
# **Clusters (without semantics)**	**MinPtsOfJ=2**	166	112	87	165	116	95	186	121	92	173	120	99
	MinPtsOfJ=4	166	112	87	165	116	95	217	147	111	207	149	112
	MinPtsOfJ=6	166	112	87	165	116	95	236	166	129	218	160	124
	MinPtsOfJ=8	166	112	87	165	116	95	248	178	141	229	171	135

Table 4.4: SEM-DTBJ-Cluster and DJ-Cluster comparison on Geolife trajectory dataset. ϵ=1 km, *MinPts*=3, *MinPtsOfJ*=2

	#Samples	Entropy	Purity	NMI	#Final Clusters
DJ-Cluster		0.4598	0.8686	0.6512	356
SEM-DTBJ-Cluster	2	0.4305	0.8743	0.6568	400
	3	0.3694	0.8852	0.6745	416
	4	0.3528	0.9190	0.6758	453
	5	0.3302	0.9305	0.6780	489

function of the parameter *MinPtsOfJ* where *MinPtsOfJ* varies from 2, to 4, 6, 8, respectively. In the scenario of Table 4.2, ϵ is set to 1 km, 2 km respectively, and *MinPts* is set to 3, 4, and 5 respectively. In Table 4.3, ϵ varies from 20 metres to 30 metres and 40 metres, *MinPts* varies from 3 to 4 and 5, and *MinPtsOfJ* varies from 2 to 4, 6, and 8. Observe that the number of clusters by using DJ-Cluster remains unchanged as long as the value of *MinPts* remains unchanged no matter how the value of *MinPtsOfJ* changes, whereas the number of clusters by using DTBJ-Cluster varies as long as the value of *MinPtsOfJ* changes.

Discussion: Merely from the comparison results of DTBJ-Cluster and DJ-Cluster, DTBJ-Cluster algorithm generates more clusters (POIs) than DJ-Cluster in that only when the intersection of two clusters contains more than *MinPtsOfJ* common points, DTBJ-Cluster merges the two clusters into one cluster. This is actually more reasonable since DTBJ-Cluster considers POIs at a finer granularity.

In order to study the performance of our proposed SEM-DTBJ-Cluster algorithm, we give the metrics and the results in Table 4.4 and Table 4.5.

Metrics: We adopt three widely used metrics, namely purity ([114]), entropy, and normalized mutual information (NMI) ([84]) to study the performance of clustering algorithms when there exists a ground truth. An algorithm with smaller entropy, or larger purity or NMI indicates that it is a better clustering algorithm.

We can find that SEM-DTBJ-Cluster outperforms DJ-Cluster, in terms of all three metrics on the two datasets. These results demonstrate that, the DTBJ-Cluster algorithm, together with the split and merge steps based on the semantics of clusters makes SEM-DTBJ-Cluster effective.

Table 4.5: SEM-DTBJ-Cluster and DJ-Cluster comparison on Illinois trajectory dataset. ϵ=30 metres, *MinPts*=4, *MinPtsOfJ*=2.

	#Samples	Entropy	Purity	NMI	#Final Clusters
DJ-Cluster		0.4501	0.8702	0.6437	116
SEM-DTBJ-Cluster	2	0.4356	0.8796	0.6599	130
	3	0.3769	0.8876	0.6731	158
	4	0.3501	0.9257	0.6756	164
	5	0.3285	0.9286	0.6769	185

4.2.4 Performance Evaluation of Our PTG-Recommend Framework

We evaluate the methods utilizing popularity influence, the methods applying temporal influence, and the methods applying geographical influence, and the final unified framework, respectively. This subsection first gives the metrics for performance evaluation, and subsequently reports the performance of the proposed methods.

4.2.4.1 Evaluation metrics

For the purpose of studying the effectiveness of our proposed methods, we adopt the following two metrics as the main measurements for the experimental evaluation. Precision, defined as P = TP/(TP+FP), is used to measure how many POIs in the first *N* recommended POIs corresponding to the number of hold-off POIs in the testing data. Recall, R = TP/(TP+FN), is a measure of the number of POIs in the hold-off POIs in the testing data selected among the first *N* recommended POIs, where TP and FP indicate the number of correct recommendations and false recommendations, and FN represents false negative recommendations , i.e., the number of POIs which are supposed to be in the set of first *N* recommended POIs but actually not in the set.

Analogously to [137], for user *u* and date *t*, in the testing data, let $R_{u,t}$ and $T_{u,t}$ denote the group of recommended POIs and the group of corresponding groundtruth POIs, respectively. We divide the POIs of the two groups into three categories and obtain the following three values: $\mathrm{TP_{u,t}}$, $\mathrm{FN_{u,t}}$, and $\mathrm{FP_{u,t}}$.

- $\mathrm{TP_{u,t}}$: The quantity of POIs belonging to both $T_{u,t}$ and $R_{u,t}$.
- $\mathrm{FN_{u,t}}$: The quantity of POIs belonging to $T_{u,t}$ but not to $R_{u,t}$.

- $\mathrm{FP_{u,t}}$: The quantity of POIs belonging to $R_{u,t}$ but not to $T_{u,t}$.

After considering the time slots (date) t, the precision and recall is calculated as follows.

$$precision(t) = \frac{\sum_{u' \in U} TP_{u',t}}{\sum_{u' \in U} (TP_{u',t} + FP_{u',t})} \tag{4.16}$$

$$recall(t) = \frac{\sum_{u' \in U} TP_{u',t}}{\sum_{u' \in U} (TP_{u',t} + TN_{u',t})} \tag{4.17}$$

The average precision and recall can be calculated by the following equations, where t denotes each date.

$$precision = \frac{1}{T} \sum_{t \in T} precision(t) \tag{4.18}$$

$$recall = \frac{1}{T} \sum_{t \in T} recall(t) \tag{4.19}$$

4.2.4.2 Methods utilizing popularity influence

Analogously, we name our proposed methods utilizing popularity influence UP for short, and abbreviate the state-of-the-art method utilizing semantic influence proposed by [132] as P. We measure the performance of these two methods by depicting the precision and recall of them as a function of the number of POIs N in the recommendation results in Fig. 4.11. We observe that the recommendation accuracy of UP is much higher than P with regard to precision and recall. Furthermore, the precision and recall of using UP method is scalable to various $\mathcal{N}$ values. This clearly illustrates that our proposed method is effective.

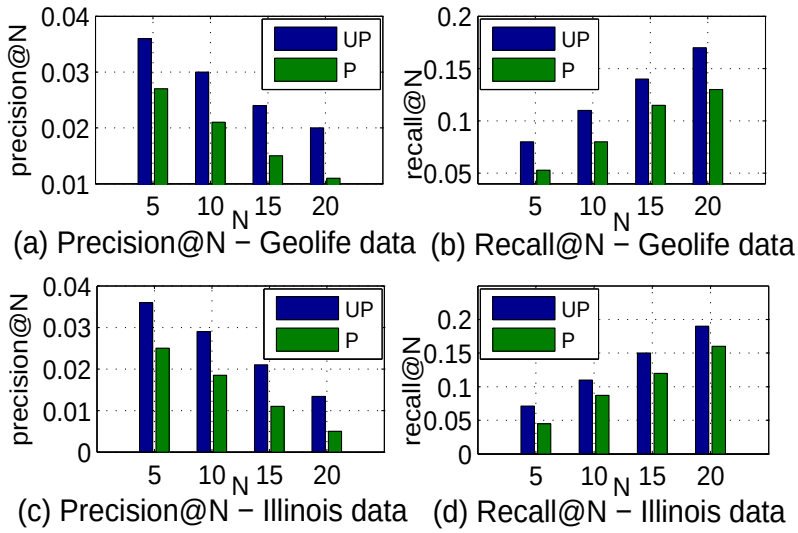


Figure 4.11: Performance of methods utilizing the influence of popularity.

4.2.4.3 Methods utilizing temporal influence

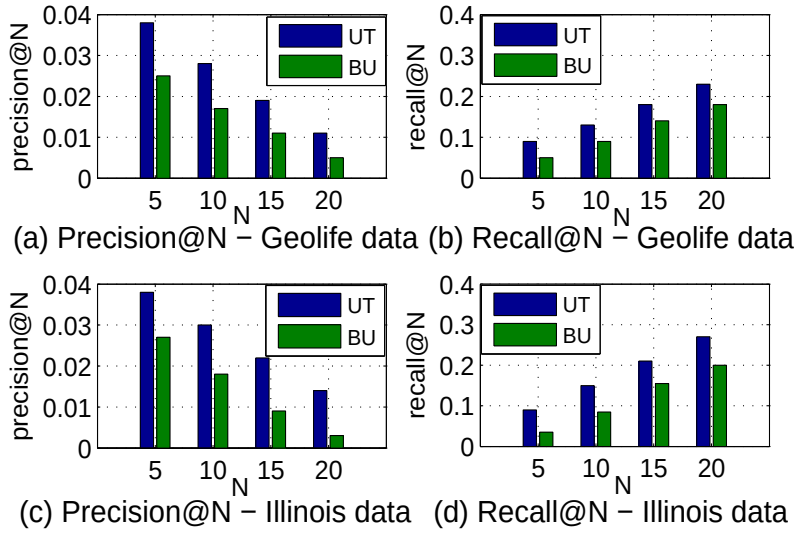


Figure 4.12: Our method applying temporal influence vs. baseline CF method.

For convenience, we name our methods utilizing temporal influence UT for short. Let BU denote the baseline user-based CF method. We compare the effectiveness of UT with BU on Geolife dataset and Illinois dataset. Figure 4.12 depicts the precision and recall of UT and BU. Observe that UT which applies the temporal influence outperforms BU, since time is not considered by BU. As for precision, UT is superior to BU on average by 20% to 30% on Geolife dataset or Illinois dataset. These results indicate that time factor is essential for POI recommendation. Among all the experiments, UT always performs

better than BU with respect to various N values on the datasets. This superior performance is since UT not only takes temporal influence into account, but also addresses the data sparsity problem by taking a further step of smoothing enhancement.

4.2.4.4 Methods utilizing geographical influence

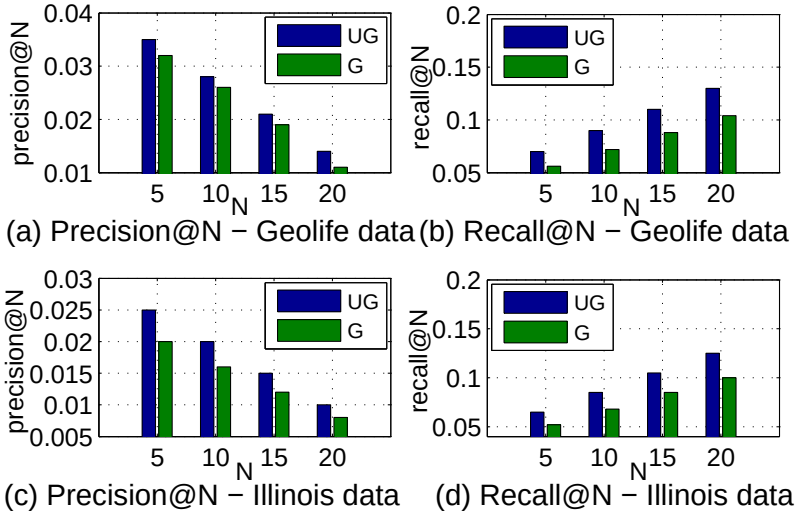
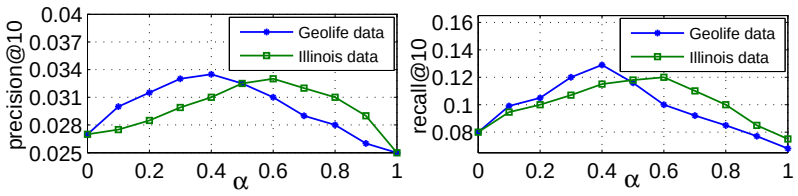


Figure 4.13: Comparison of methods utilizing geographical influence.

With the aim of comparing our proposed methods utilizing geographical influence (named UG for short) with the baseline method (denoted as G) proposed by [129], we plot the precision and recall of UG and G as a function of the number of the top recommended N POIs respectively in Fig. 4.13. Observe that UG performs better than G on either Geolife dataset or Illinois dataset, in terms of precision and recall. At the same time, the precision and recall of UG is scalable to different N values. These results show that our methods utilizing geographical influence are more effective than the baseline method G.

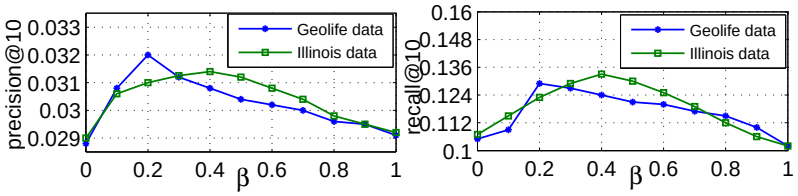
4.2.4.5 Unified framework

Finally, we evaluate our unified framework PTG-Recommend. For PTG-Recommend, two parameters α and β are used to tune the weights of the three parts in Eq. 4.15. We tune α and β separately, and plot the average precision and recall with respect to different α and β, respectively. As depicted in Fig. 4.14, we set β to a fixed value $\beta = 0.2$, and the optimal precision and recall can be achieved by setting $\alpha = 0.4$ and 0.6 on Geolife dataset and Illinois dataset, respectively. Likewise, we set $\alpha = 0.4$ in Fig. 4.15, and the



(a) Precision@10 (b) Recall@10

Figure 4.14: Tuning parameter α.



(a) Precision@10 (b) Recall@10

Figure 4.15: Tuning parameter β.

optimal precision and recall can be achieved if β equals about 0.2 and 0.4 on Geolife dataset and Illinois dataset, respectively. We subsequently compare PTG-Recommend with the above three methods. Without loss of generality, we set α = 0.4, and β = 0.2 and 0.4 for Geolife dataset and Illinois dataset, respectively. As shown in Fig. 4.16, PTG-Recommend accomplishes the best performance with respect to precision and recall. The results demonstrate that our framework makes good use of semantics, popularity, temporal, and geographical effects and outperforms the best method available in literature with regard to effectiveness.

Discussion on precision and recall. Observe that the precision and recall shown in the above figures are a bit low. The evaluation is to compare the relative performance of the different methods. actually, it is possible that a recommended POI are interesting to a user, but it does not appear in testing data. So in other words, the precision/recall is higher in practice. Almost all the recommendation problems have the same problem for evaluation.

4.3 Conclusions

In this chapter, we develop a semantic-temporal-geographical framework, namely, PTG-Recommend, for recommending POIs to a user from users' GPS trajectories. Our framework firstly develops a SEM-DTBJ-Cluster algorithm, which is a novel semantics-enhanced

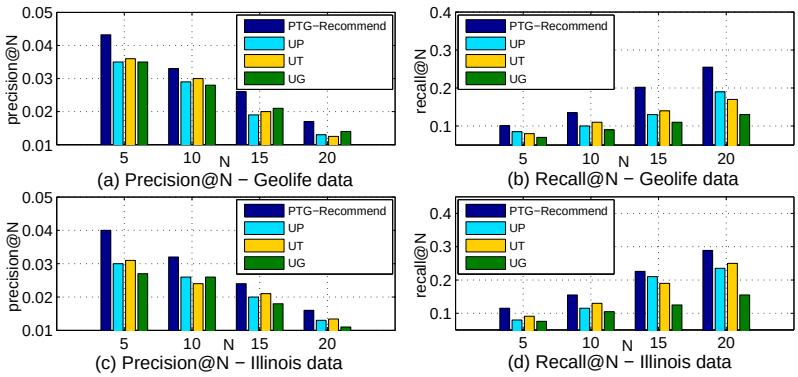


Figure 4.16: Performance of the unified framework.

clustering algorithm, to extract semantic POIs from GPS data. Our framework then takes the influence of popularity, the influence of temporal features, and the influence of geographical features of each POI into consideration, and derives a popularity scoring function, a temporal scoring function, and a geographical scoring function, respectively. Finally, our framework combines the above three functions together and obtains a unified recommendation score for each POI for a user. To the best of our knowledge, our framework is the first one that recommends POIs by exploiting the popularity, temporal information, and geographical information of trajectories. We carry out experiments to evaluate the proposed methods respectively. Experimental results demonstrate that the PTG-Recommend framework outperforms the baseline methods with regard to recommendation precision and recall by 20% to 30%.

Chapter 5

Cost-Optimal Route Search in Time-Dependent Road Networks

This chapter solves the problem of cost-optimal time-dependent routing (COTER). Section 5.1 models the time-dependent road network, defines the cost-optimal route query problem in detail, and analyzes fuel consumption model, travel time function, as well as the toll fee function. We propose our approximate algorithm, namely, ALG-COTER, to answer the COTER query in Section 5.2. We subsequently carry out several groups of experiments to evaluate our algorithms and report the experimental results in Section 5.3. We finally summarize this chapter in Section 5.4.

5.1 Problem Formulation

5.1.1 Problem Setting and Definitions

Definition 15 ***Time-dependent road network****: A time-dependent road network* G_T = (V, E, L, W, C, F)*, where* V = $\{n_i\}$ *represents a set of nodes,* $E \subseteq V \times V$ *represents a set of edges,* L *is the set of lengths of edges, and* W*,* C *and* F *are three sets of time-dependent functions. Every edge* e = (n_i, n_j) $\in E$ *has four functions: the length* $len(e)$ $\in L$*, travel time function* $w_{i,j}(T, \mathbf{v}) \in W$*, fuel consumption* $c_{i,j}(T, \mathbf{v}) \in C$*, and toll fee function* $f_{i,j}(T) \in F$*, where the time variable* T *denotes the departure time from* n_i*. Note that the length of an edge is a fixed value which is independent of time.* $w_{i,j}(T, \mathbf{v})$ *specifies how much time is needed to traverse (*n_i*,* n_j*), if departing from* n_i *at time instance* T*.* $c_{i,j}(T, \mathbf{v})$ *specifies the fuel cost for traversing (*n_i*,* n_j*) if departing from* n_i *at time instance* T*. Note that* $\mathbf{v}$ *refers to a row vector containing average speeds on edge* e *during each*

time interval. $f_{i,j}(T)$ specifies how much toll fee it costs to traverse (n_i, n_j), if departing from n_i at time instance T. ♡

We assume that $w_{i,j}(T, \mathbf{v}) \geq 0$, $c_{i,j}(T, \mathbf{v}) \geq 0$, and $f_{i,j}(T) \geq 0$. In G_T, for each edge (n_i, n_j), $w_{i,j}(T, \mathbf{v})$, $c_{i,j}(T, \mathbf{v})$, and $f_{i,j}(T)$ are all dependent on the departure time T.

Definition 16 ***Time-dependent maximum speed****: The time-dependent maximum speed $v_{max}(e, t)$ for edge e is a piecewise constant function of time $t \in (I_0, I_p]$ where p denotes the number of piecewise time intervals. It is formally defined as:*

$$v_{max}(e,t) = \begin{cases} v_0; & t \in (I_0, I_1] \\ v_1; & t \in (I_1, I_2] \\ \cdots & \\ v_{p-1}; & t \in (I_{p-1}, I_p] \end{cases} \tag{5.1}$$

where v_i denotes the maximum speed allowed on edge e during time interval (I_i, I_{i+1}] for $i \in [0, p-1]$. ♡

Note that the time period t is different from the departure time T. $v_{max}(e,t)$ gives an upper bound of the speeds on an edge during different time periods. We assume that among all the edges, the maximum value and the minimum value of $v_{max}(e,t)$ are 130 $\mathrm{km/h}$ denoted as V_{max}, and 40 $\mathrm{km/h}$ denoted as V_{min}, respectively.

Definition 17 ***Waiting time****: We allow some waiting time (measured by non-negative integers) at some nodes. Let $\gamma(n_i)$ denote the waiting time at node n_i. Let V_w and V_{nw} denote the set of nodes which allow waiting, and the set of nodes which disallow waiting, respectively ($V_w \cap V_{nw} = \emptyset$ and $V_w \cup V_{nw} = V$).* ♡

If $n_i \in V_w$, then waiting is allowed at node n_i which means $\gamma(n_i) \geq 0$ and meanwhile $\gamma(n_i) \in \mathbf{N}$ where $\mathbf{N}$ denotes the set of integers which are not smaller than 0; otherwise, if $n_i \in V_{nw}$, then no waiting is allowed at n_i, i.e., $\gamma(n_i)$ strictly equals 0.

Definition 18 ***Arrival (Departure) time****: The arrival time at node n_i and the departure time from n_i are denoted by $\mathsf{Arr}(n_i)$ and $\mathsf{Dep}(n_i)$, respectively. Then we have the equation below:*

$$\mathsf{Dep}(n_i) = \mathsf{Arr}(n_i) + \gamma(n_i)$$

Let $R = n_1 \to n_2 \to \cdots \to n_h$ be a given route. The earliest departure time from n_1 is t_d, and we have

$$\begin{aligned}
&\mathsf{Arr}(n_1) = t_d \\
&\cdots \\
&\mathsf{Dep}(n_{h-1}) = \mathsf{Arr}(n_{h-1}) + \gamma(n_{h-1}) \\
&\mathsf{Arr}(n_h) = \mathsf{Dep}(n_{h-1}) + w_{h-1,h}(\mathsf{Dep}(n_{h-1}))
\end{aligned} \quad \heartsuit$$

Definition 19 ***(Travel) cost:*** *The (travel) cost of a route refers to the expenses (fuel expense plus toll fee) of this route. Given a route $R = n_1 \rightarrow n_2 \rightarrow \cdots \rightarrow n_h$, for any node $n_i \in R$, let $\mathsf{cost}_R(n_i)$ denote the cost from n_1 to n_i by route R. $\mathsf{cost}_R(n_i)$ is as follows:*

$$\begin{aligned}
&\mathsf{cost}_R(n_1) = 0 \\
&\mathsf{cost}_R(n_h) = \sum_{i=1}^{h-1}[f_{i,i+1}(\mathsf{Dep}(n_i)) + c_{i,i+1}(\mathsf{Dep}(n_i))]
\end{aligned}$$

where $f_{i,i+1}(\mathsf{Dep}(n_i))$ denotes the toll fee of edge (n_i, n_{i+1}) if leaving n_i at time $\mathsf{Dep}(n_i)$, and $c_{i,i+1}(\mathsf{Dep}(n_i))$ denotes the fuel cost of edge (n_i, n_{i+1}) if leaving n_i at time $\mathsf{Dep}(n_i)$. The cost of route R is defined as $\mathsf{cost}(R) = \mathsf{cost}_R(n_h)$. Let $c(R)$ and $f(R)$ denote the fuel cost, and the toll fee of R, then $\mathsf{cost}(R) = c(R) + f(R)$. $\heartsuit$

Definition 20 COTER ***(Cost-Optimal Time-dEpendent Routing)****: Given a time-dependent road network G_T, a start node n_s, an end node n_e, an earliest departure time stamp t_d, and a latest arrival time stamp t_a, the* COTER *query, denoted as $\langle G_T, n_s, n_e, t_d, t_a \rangle$, aims to find the cost-optimal route R such that*

$$\begin{aligned}
& R = \mathrm{argmin}_R \mathsf{cost}(R) \\
\textit{subject to} \quad & \mathsf{Dep}(n_s) \geq t_d \\
& \mathsf{Arr}(n_e) \leq t_a \\
& \gamma(n_i) = 0, \textit{ for each } n_i \in V_{nw} \\
& \gamma(n_j) \in \mathbf{N}, \textit{ for each } n_j \in V_w \\
& v(e,t) \leq v_{max}(e,t), \textit{ where } t \in [t_d, t_a] \textit{ for any } e \in R
\end{aligned} \quad \heartsuit$$

COTER queries a route R which starts at n_s and ends at n_e, such that R minimizes $\mathsf{cost}(R)$ with (i) time constraints: $(\mathsf{Dep}(n_s) \geq t_d) \wedge (\mathsf{Arr}(n_e) \leq t_a) \wedge (\gamma(n_i) = 0$ for $n_i \in V_{nw}) \wedge (\gamma(n_j) \in \mathbf{N}$ for $n_j \in V_w)$; and (ii) speed constraints: the average speed $v(e,t)$ on each edge $e \in R$ during each time interval should be no larger than the maximum speed allowed during each corresponding time interval.

Theorem 5.1. *The problem of solving* COTER *queries is NP-hard.* $\spadesuit$

PROOF. From [86], we know that resource constrained shortest path problem (denoted as CSP) is NP-complete even for the case of one resource. CSP asks for the computation

of a least cost path obeying a set of resource constraints. The problem of COTER can be regarded as an extension of the NP-hard CSP. Our COTER problem has three kinds of constraints: (1) the travel time budget constraint; (2) the speed constraint; (3) the waiting time constraint. If we disregard the speed constraint as well as the waiting time constraint, the problem of solving COTER becomes CSP with one resource constraint. ■

Definition 21 ***Feasible arrival time interval****: The feasible arrival time interval of a node n_i is an interval of time stamps at which one possibly arrives at node n_i, satisfying* $\mathsf{Dep}(n_s) \geq t_d$ *and* $\mathsf{Arr}(n_e) \leq t_a$*. Let λ_i denote the earliest arrival time at node n_i if departing n_s at or after time t_d, and θ_i denote the latest arrival time at node n_i if a user wants to arrive at the destination node n_e before or at time t_a by passing n_i. Then the feasible arrival time interval of n_i is [λ_i, θ_i].* ♡

Definition 22 ***Candidate node****: A candidate node is a node n_i whose feasible arrival time interval [λ_i, θ_i] is within the entire time interval [t_d, t_a], because only these nodes can be reached from n_s and meanwhile can reach n_e under the given time constraints.* ♡

Notations. We give the notations in Table 5.1.

5.1.2 Fuel Consumption and Travel Time Functions

The fuel consumption and travel time on edge (n_i, n_j) are dependent on the departure time T from node n_i and the driving speeds.

5.1.2.1 Fuel consumption models

According to the previous study [51], it is suggested that the SIDRA-Avg model can only be used in urban road networks and that the average travel speed should be below a certain threshold v^* (usually $v^* \geq$ 50km/h), e.g., v^*=50km/h. When the average speed exceeds v^*, the SIDRA-Running model should be used instead.

(i) **SIDRA-Avg.** The fuel consumption per unit distance $f_a(v_s)$ (mL/km) of the SIDRA-Avg model is defined as follows.

$$f_a(v_s) = \frac{1600}{v_s} + 73.8, \tag{5.2}$$

where v_s is the average travel speed (km/h).

(ii) **SIDRA-Running.** The fuel consumption of the running mode F_s (mL) is estimated as

$$F_s = F_{idle} + f_r(v_r) \cdot x_s, \tag{5.3}$$

where $F_{idle} = 0.444 \cdot t_{idle}$ is the fuel consumption during idle periods and $f_r(v_r)$ (mL/km) indicates the average fuel consumption per unit distance excluding idle periods, which is computed as follows.

$$\begin{aligned} f_r(v_r) = & \frac{1600}{v_r} + 30 + 0.0075 \cdot v_r^2 + 108 \cdot k_{E1} \cdot E_{k+} \\ & + 54 \cdot k_{E2} \cdot E_{K+}^2 + 10.6 \cdot k_G \cdot \theta \end{aligned} \tag{5.4}$$

where v_r is the average running speed (km/h), x_s (km) is the total travel distance; θ (%) is the road grade; E_{k+} is the marginal fuel consumption due to speed fluctuations; and k_{E1}, k_{E2}, and k_G are calibration parameters.

Let $len(e)$ denote the length of an edge $e = (n_i, n_j)$. Suppose the fuel price is $\$a_5$ per milliliter (in our experiments, we set a_5 = 0.00054155), and the average speed on edge (n_i, n_j) is $\bar{v}$, then we derive the fuel cost $c_{i,j}$ on edge (n_i, n_j) as Eq. 5.5.

$$c_{i,j} = \begin{cases} f_a(\bar{v}) \cdot len(e) \cdot a_5, & \bar{v} < v^* \\ (f_r(\bar{v}) \cdot len(e) + F_{idle}) \cdot a_5, & \bar{v} \geq v^* \end{cases} \tag{5.5}$$

We compute the derivative of $c_{i,j}$ as follows:

(i) $\bar{v} < v^*$

$$\frac{d(c_{i,j})}{d\bar{v}} = len(e) \cdot a_5 \cdot (-\frac{1600}{\bar{v}^2}) < 0$$

Hence, when $\bar{v} < v^*$ km/h, the larger the average speed $\bar{v}$ is, the less fuel cost is consumed.

(ii) $\bar{v} \geq v^*$

$$\frac{d(c_{i,j})}{d\bar{v}} = len(e) \cdot a_5 \cdot (-\frac{1600}{\bar{v}^2} + 2 * 0.0075 \cdot \bar{v})$$

When $\bar{v} > 47.43$, we find that $\frac{d(c_{i,j})}{d\bar{v}} > 0$. Hence, when $\bar{v} \geq v^*$, the smaller the average speed $\bar{v}$ is, the less fuel cost is consumed and the minimum fuel cost is achieved by setting $v = v^*$ for $\bar{v} \geq 50$ km/h.

To minimize the fuel consumption, the average speed $\bar{v}$ in Eq. 5.5 should be:

$$\bar{v} = \begin{cases} v^*; & \text{if } v_{max}(e,t) \geq v^* \\ v_{max}(e,t); & \text{if } v_{max}(e,t) < v^* \end{cases}$$

5.1.2.2 Fuel cost minimization for an edge when travel time is fixed

For an edge e = (n_i, n_j), its maximum speed function $v_{max}(e,t)$ is given as Eq. 5.1. Suppose the departure time T falls within the k-th time interval, i.e., $T \in$ (I_k, I_{k+1}] where $0 \leq k \leq p-1$, if the travel time on e is w, then obviously the arrival time at n_j should be $T + w$. Suppose $T + w$ falls within the m-th time interval, i.e. $T + w \in (I_m, I_{m+1}]$, then $k \leq m \leq p-1$, and thus (i) the whole edge is split into $(m - k + 1)$ line segments; (ii) travel time w on this edge is split into $(m - k+1)$ portions, each of which is denoted as each element of a column vector $\mathbf{t}$ in Eq. 5.6; (iii) the corresponding departure time for each time interval of $\mathbf{t}$ is denoted as each element of a column vector $\mathbf{T}$ in Eq. 5.7; and (iv) the corresponding average speed during each time interval is denoted as each element of a row vector $\mathbf{v}$ in Eq. 5.8.

From Eq. 5.6 and Eq. 5.7, we see that

- If m equals k, then both the departure time T and the arrival time $T + w$ fall within the same time interval (I_k, I_{k+1}], so we have $I_k \leq T \leq T + w \leq I_{k+1}$ within only one time interval and the travel time is w;
- If $m = k + 1$, then we have $I_k \leq T \leq I_{k+1} \leq T + w \leq I_{m+1}$, so the travel time is split into two time intervals, $[T, I_{k+1}]$ and $(I_{k+1}, T + w]$. The corresponding departure time for these two time intervals is T and I_{k+1}, respectively.
- If $m > k + 1$, then we have $I_k \leq T \leq I_{k+1} \leq \cdots \leq I_m \leq T + w \leq I_{m+1}$, so the travel time is split into $(m - k + 1)$ time intervals. The corresponding departure time for the $(m - k + 1)$ time intervals is T, I_{k+1}, $\cdots$, I_m, respectively.

$$\mathbf{t} = \begin{cases} [w] & \textit{if } m = k \\ [I_{k+1} - T, T + w - I_{k+1}]^T & \textit{if } m = k + 1 \\ [I_{k+1} - T, I_{k+2} - I_{k+1}, \cdots, T + w - I_m]^T & \textit{if } m > k + 1 \end{cases} \tag{5.6}$$

$$\mathbf{T} = \begin{cases} [T] & \textit{if } m = k \\ [T, I_{k+1}]^T & \textit{if } m = k+1 \\ [T, I_{k+1}, \cdots, I_m]^T & \textit{if } m > k+1 \end{cases} \tag{5.7}$$

$$\mathbf{v} = \begin{cases} [\bar{v}_k] & \textit{if } m = k \\ [\bar{v}_k, \bar{v}_{k+1}] & \textit{if } m = k+1 \\ [\bar{v}_k, \bar{v}_{k+1}, \cdots, \bar{v}_m] & \textit{if } m > k+1 \end{cases} \tag{5.8}$$

where, $\bar{v}_i$ ($k \leq i \leq m$) refers to the average speed during time interval (I_i, I_{i+1}] on edge e, and $\bar{v}_i \leq v_{max}(e, (I_i, I_{i+1}]) = v_i$.

Meanwhile, the edge e is split into ($m - k$+1) segments. The length of each segment is equal to the corresponding average speed multiplied by the corresponding travel time during each time interval.

$$len(e) = len_1 + \cdots + len_{m-k+1} = \mathbf{v} \cdot \mathbf{t}$$

Then, with the departure time T and travel time w, the fuel cost on edge e, $c_{i,j}(T, \mathbf{v})$, is computed as Eq. 5.9:

$$c_{i,j}(T, \mathbf{v}) = \sum_{n=1}^{m-k+1} c_n(\mathbf{T}_{n,1}, \mathbf{v}_{1,n}) \tag{5.9}$$

where, $\forall\ n \in$ [1,m-k+1]:

$$\begin{aligned} & c_n(\mathbf{T}_{n,1}, \mathbf{v}_{1,n}) \\ = & \begin{cases} f_a(\mathbf{v}_{1,n}) \cdot (\mathbf{v}_{1,n} \cdot \mathbf{t}_{n,1}) \cdot a_5, & \mathbf{v}_{1,n} < 50 \\ (f_r(\mathbf{v}_{1,n}) \cdot (\mathbf{v}_{1,n} \cdot \mathbf{t}_{n,1}) + F_i) \cdot a_5, & \mathbf{v}_{1,n} \geq 50 \end{cases} \end{aligned} \tag{5.10}$$

where, $\mathbf{T}_{n,1}$ and $\mathbf{t}_{n,1}$ refer to the element in the n-th row and 1st column of $\mathbf{T}$ and $\mathbf{t}$; $\mathbf{v}_{1,n}$ refers to the element in the 1st row and n-th column of $\mathbf{v}$.

In order to minimize the fuel cost under the constraint that the travel time is exactly w, we formulate this **nonlinear programming optimization** problem as follows:

$$\begin{aligned} & \min_{\mathbf{v}} \quad c_{i,j}(T, \mathbf{v}) \\ \textit{subject to} \quad & w_{i,j}(T, \mathbf{v}) = w \\ & \mathbf{v} \cdot \mathbf{t} = len(e) \\ & \bar{v}_k \leq v_k \\ & \cdots \\ & \bar{v}_m \leq v_m \end{aligned} \tag{5.11}$$

Objective function in Eq. 5.11 minimizes the fuel cost of edge e=(n_i, n_j) with given departure time T. Constraints in Eq. 5.11 include (i) the sum of the $(m-k+1)$ travel-time intervals equals the total travel time w; (ii) the sum of the distances of $(m-k+1)$ time intervals equals the total length $len(e)$; (iii) $\forall n \in [1, m-k+1]$: the average speed $\mathbf{v}_{1,n}$ = $\bar{v}_n$ of each time interval $(I_{k+n-1}, I_{k+n}]$ should satisfy the maximum speed allowed on edge e during the current time interval.

Algorithm 1: Compute-Minimum-Fuel-Cost (e, T, w, $v_{max}(e,t)$, t_k, t_m)

Input: e = (n_i, n_j), T, w, $v_{max}(e,t)$, t_k, t_m
Output: a quadruple q = $\langle c_{i,j}(T, \mathbf{v}), w, T, \mathbf{v} \rangle$
1 solve the nonlinear optimization problem in Eq. 11;
2 **return** quadruple q = $\langle c_{i,j}(T, \mathbf{v}), w, T, \mathbf{v} \rangle$;

Figure 5.1: Algorithm 1: Compute-Minimum-Fuel-Cost.

Figure 5.1 describes Algorithm 1 (Compute-Minimum-Fuel-Cost). Algorithm 1 solves the nonlinear programming problem of minimizing fuel cost of edge e under the constraint that the travel time is exactly w by using the genetic algorithm. After retrieving the optimal $\mathbf{v}$ which leads to the minimum $c_{i,j}(T, \mathbf{v})$, Algorithm 1 returns a quadruple encapsulating the minimum fuel consumption, the corresponding travel time w, departure time T, and speed $\mathbf{v}$.

5.1.3 Toll Fee Functions

In our setting, the toll fee function $f_{i,j}(T)$ of edge (n_i, n_j) is an arbitrary single-valued function of the departure time T, and our algorithm could solve COTER with arbitrary

Algorithm 2: ALG-COTER(G_T, n_s, n_e, t_d, t_a)

Input: G_T, n_s, n_e, t_d, t_a
Output: optimal route R from n_s to n_e

```
1  // Step 1:
2  COMPUTE-EARLIEST-ARRIVAL-TIME(G_T, n_s);
3
4  // Step 2:
5  G_T^c ← the converse graph of G_T;
6  COMPUTE-LATEST-ARRIVAL-TIME(G_T^c, n_e);
7
8  // Step 3:
9  Vec ← an empty vector that will contain the sorted nodes;
10 |V| ← the number of nodes in G_T;
11 bool MarkByTopo[|V|];
12 memset(MarkByTopo, 0, sizeof(MarkByTopo));
13 TOPOLOGICAL-SORT(n_e, MarkByTopo);
14
15 // Step 4:
16 COMPUTE-MINIMUM-COST(Vec, t_d, t_a);
17
18 // Step 5:
19 if τ_e < ∞ then
20     BACKTRACK-OPTIMAL-ROUTE(G_T, g_e(t_e), t_e);
```

Figure 5.2: Algorithm 2: ALG-COTER algorithm.

single-valued toll fee functions.

5.2 Algorithms

As the problem of COTER is NP-hard, we propose our approximate ALG-COTER algorithm for tackling COTER and analyze its time complexity in this section.

Our ALG-COTER algorithm has five steps and is shown in Fig. 5.2. Step 1 is computing the earliest arrival time λ_i for each node n_i if departing from the source node n_s at t_d. Step 2 is computing θ_i, which is the latest feasible arrival time at n_i if one wants to arrive at n_e before or at time t_a, for each candidate node n_i. Step 3 is getting a topological sort of the candidate nodes. Step 4 is computing the minimum cost when arriving at each candidate node n_i at different arrival time by recursively computing the OC-functions according to the topological order. Step 5 is backtracking the cost-optimal route R and finding the waiting time at each node together with the speeds on each edge in the optimal route R.

5.2.1 Computing the Earliest Arrival Time λ_i for Each Descendant Node of n_s

We present the first step of the ALG-COTER, i.e., computing the earliest arrival time λ_i for descendant nodes of n_s.

Suppose the earliest departure time $\lambda_j \in (t_k, t_{k+1}]$ from n_j for edge $e = (n_j, n_i)$ is given, then if a user always traverses edge e with the maximum speed allowed on e, we can obtain the minimum travel time for the specific departure time λ_j for e, i.e., $\min_{\mathbf{v}}\{w_{j,i}(\lambda_j, \mathbf{v})\}$, and $\lambda_j + \min_{\mathbf{v}}\{w_{j,i}(\lambda_j, \mathbf{v})\}$ is the earliest arrival time at n_i.

Initially, for each node $n_i \in V$, we set $\lambda_i = \infty$. For the source node n_s, λ_s is set as t_d, then we easily compute the earliest arrival time at each of the outgoing neighbors of n_s. Analogously, the earliest arrival time at every descendant node n_i of n_s can be obtained by performing the time-dependent single-source shortest path algorithm on G_T from n_s.

Obviously, a time-dependent Fibonacci-heap optimized Dijkstra's algorithm is enough. (According to [107], this combination is still the fastest known algorithm for solving the SSSP problem with non-negative real edge weights.) The worst time complexity is $O(|V| \log |V| + |E|)$ (in the worst case, the whole road network is involved), where $|V|$ and $|E|$ represent the number of nodes and number of edges in G_T.

5.2.2 Computing the Latest Arrival Time θ_i for Candidate Nodes

We now present Step 2 of our ALG-COTER, i.e., computing the latest arrival time θ_i for each candidate node n_i.

Let G_T^c denote the converse graph of G_T. In G_T^c, the source node is n_e and $\theta_e = t_a$. The shortest travel time of the edge in G_T^c can be easily obtained by setting the speed on this edge as the upperbound of maximum speeds V_{max}. Then the edge weight on edge $e = (n_j, n_i)$ in G_T^c is equal to $-\frac{len(e)}{V_{max}}$. Hence, for edge e and node n_i, the latest arrival time $\theta_i = \max\{\theta_i, \theta_j - \frac{len(e)}{V_{max}}\}$. In other words, this problem is adapted to the single-source longest path problem with negative weights which can also be solved by Fibonacci-heap optimized Dijkstra's algorithm. The worst time complexity of this algorithm is $O(|V| \log |V| + |E|)$.

5.2.3 Topologically Sorting Candidate Nodes

The algorithm for Step 3 is named as TOPOLOGICAL-SORT (Algorithm 3), and is shown in Fig. 5.3.

Algorithm 3: TOPOLOGICAL-SORT(n_i, *MarkByTopo*)

Input: n_i, *MarkByTopo*
Output: a vector *Vec* of all the nodes on any route from n_s to n_e in topological order

```
1 MarkByTopo[n_i] ← 1;
2 for each node n_j ∈ N^-(n_i) do
3     if MarkByTopo[n_j] ≠ 1 and λ_j ≥ t_d and λ_j ≤ t_a and θ_j ≥ t_d and θ_j ≤ t_a then
4         TOPOLOGICAL-SORT(n_j, MarkByTopo);
5 Vec.push_back(n_i);
```

Figure 5.3: Algorithm 3: TOPOLOGICAL-SORT (Step 3).

In Algorithm 3, topological sorting is designed as a recursive function based on depth first search. *MarkByTopo*[n_i] = 0 means the node n_i has not been traversed; and *MarkByTopo*[n_i] = 1 means the node n_i has been traversed already. If node n_i has not been traversed, i.e., *MarkByTopo*[n_i] = 0, then the algorithm traverses n_i. Then for each of n_i's predecessor node n_j, if n_j has not been traversed, and meanwhile its arrival time interval is within the entire time interval [t_d, t_a], then the algorithm traverses n_j (Lines 3 - 4). Line 3 of TOPOLOGICAL-SORT guarantees that we only sort candidate nodes, namely, those nodes whose feasible arrival time interval is within the entire time interval [t_d, t_a], because only these nodes can be reached from n_s and meanwhile can reach n_e under the given time constraints. Finally, after all of n_i's predecessor nodes have been traversed, which means after all of n_i's predecessor nodes have been added into the vector *Vec*, the algorithm adds n_i to the vector *Vec* (Line 5). This guarantees that all of n_i's predecessor nodes locate in front of n_i in the vector *Vec*. The time complexity of the TOPOLOGICAL-SORT in the worst case is the same as the time complexity of a plain depth first search in the worst case, i.e., $O(|V| + |E|)$.

The topological order is the input of the fourth step detailed in Section 5.2.4. The fourth step is recursively computing the values of the optimal-cost (OC) function at each node according to the recurrence relation formula between the OC-functions of each node and the OC-functions of its predecessor nodes. The topological order obtained by Step

3, guarantees that, the minimum cost at a node n_i which locates in the topological order, can be computed by the minimum cost at those nodes which locate in front of n_i in the topological order. The topological order also guarantees that the OC-function values of n_i's ancestor nodes are computed before we compute n_i's OC-function value.

5.2.4 Computing the Minimum Cost

We now details the fourth step of our ALG-COTER algorithm, i.e., computing the minimum cost for the candidate nodes by computing the OC-functions according to the topological order. The algorithm for Step 4 is named as COMPUTE-MINIMUM-COST (Algorithm 4) and shown in Fig. 5.4.

5.2.4.1 Optimal cost function $opt_i(t)$

We introduce the "Optimal Cost when arriving at t function", or OC-function for short, $opt_i(t)$, for each candidate node $n_i \in G_T$. For each candidate node n_i, we use $R_i(t)$ to denote the set of all the routes which departs from the source n_s at or after time t_d and arrives at n_i at time instance t.

The data structure of $opt_i(t)$ is a quadruple which encapsulates: (i) the value of optimal (minimum) cost $opt_i(t).val$ when arriving at n_i at time t, that is, $opt_i(t).val = \min\{\mathsf{cost}(r)|r \in R_i(t)\}$; (ii) the previous node $opt_i(t).pre$ which is the very incoming neighbor of n_i from which the minimum cost $opt_i(t).val$ can be obtained; (iii) the corresponding quadruple $opt_i(t).q$ which contains the corresponding fuel consumption cost, the travel time, the departure time T from $opt_i(t).pre$, and speed $\mathbf{v}$; (iv) and the previous minimum cost $opt_i(t).\textit{preCost}$ when arriving at the previous node $opt_i(t).pre$ before or at time instance T.

$opt_e(t_e)$ is the quadruple to encapsulate the minimum value of $opt_e(t).val$ at destination n_e, where t_e is the time instance at which $opt_e(t).val$ is minimized. Spontaneously, $opt_e(t_e).val$ is the optimal cost from n_s to n_e under the constraints. Our ultimate goal is to obtain $opt_e(t_e)$.

5.2.4.2 Recurrence relation formula of OC-functions

Suppose node n_j is one incoming neighbor of node n_i, i.e., $n_j \in N^-(n_i)$, then the recurrence relation between n_i's OC-functions and n_j's OC-functions can be expressed by Eq. 5.12. The key idea of COMPUTE-MINIMUM-COST (Algorithm 4) is to use Eq. 5.12 to compute the value of the OC-function $opt_i(t_i).val$ recursively.

$$opt_i(t_i).val = opt_j(t_j).val + f_{j,i}(opt_i(t_i).q.T) + c_{j,i}(opt_i(t_i).q.T, \mathbf{v}) \tag{5.12}$$

5.2.4.3 COMPUTE-MINIMUM-COST algorithm

COMPUTE-MINIMUM-COST algorithm computes $opt_e(t_e)$ iteratively. This algorithm uses the output of the previous three steps.

In Fig. 5.4, it is straightforward that the first element of *Vec* is the source node n_s and the last element of *Vec* is the destination node n_e. Initially (Lines 1 - 5), for any node n_i, $opt_i(t)$, and τ_i are initialized as $opt_i(t).val \leftarrow \infty$, and $\tau_i \leftarrow \infty$. Then a loop (Lines 6 - 35) is performed for each node in the topologically sorted candidate nodes vector *Vec*.

In the first iteration (Lines 7 - 13), *Vec*[0] is n_s, the time domain for n_s is $[\lambda_s, \theta_s]$, $opt_s(t)$ and τ_s are set as $opt_s(t).val \leftarrow 0$, $opt_s(t).pre \leftarrow$ *NULL*, $opt_s(t).q \leftarrow$ *NULL*, $opt_s(t).preCost \leftarrow 0$, and $\tau_s \leftarrow 0$, respectively. Obviously, the cost from n_s to n_s is zero for any arrival time t. It means the OC-function $opt_s(t).val$ of n_s is zero for any $t \in [\lambda_s, \theta_s]$.

In each of the subsequent iterations (Lines 14 - 35), suppose the current node *Vec*$[it]$ is n_i. Then, for each incoming neighbor n_j of n_i, if $t_d \leq \lambda_j \leq t_a$ && $\lambda_j \leq \theta_j \leq t_a$ (Line 17), then n_j must be a candidate node, which is reachable from n_s and can reach n_e under the given time constraint, and which is thus definitely in the vector *Vec*. Additionally, according to the topological order, n_j must have been already processed before n_i. Let T denote the departure time from n_j. The time domain of T is [λ_j, θ_j].

Note that waiting is allowed only at some nodes. Then, for each time stamp T (Line

Algorithm 4: COMPUTE-MINIMUM-COST(Vec, t_d, t_a)

Input: Vec, t_d, t_a
Output: OC function $opt_e(t_e)$ from n_s to n_e

for $it \in$ *[0, Vec.size() - 1]* **do**
  $n_i \leftarrow Vec[it]$;
  for $t \in [\lambda_i, \theta_i]$ **do**
    $opt_i(t).val \leftarrow \infty$;
  $\tau_i \leftarrow \infty$;
for $it \in$ *[0, Vec.size() - 1]* **do**
  if $Vec[it] = n_s$ **then**
    $opt_s(t).val \leftarrow 0$ for $t \in [\lambda_s, \theta_s]$;
    $opt_s(t).pre \leftarrow NULL$ for $t \in [\lambda_s, \theta_s]$;
    $opt_s(t).q \leftarrow NULL$ for $t \in [\lambda_s, \theta_s]$;
    $opt_s(t).preCost \leftarrow 0$ for $t \in [\lambda_s, \theta_s]$;
    $\tau_s \leftarrow 0$;
    $\mu_s \leftarrow \lambda_s$;
  else
    $n_i \leftarrow Vec[it]$;
    for *each node* $n_j \in N^-(n_i)$ **do**
      if $\lambda_j \leq \theta_j$ *and* $\lambda_j \geq t_d$ *and* $\theta_j \leq t_a$ *and* $\theta_j \geq t_d$ **then**
        `// Make sure` $n_j \in$ `Vec, and` n_j `is before` n_i `in Vec`
        for *each departure time instance* $T \in [\lambda_j, \theta_j]$ *from* n_j **do**
          if *waiting is not allowed at* n_j **then**
            $\tau_j \leftarrow opt_j(T).val$;
          else if *waiting is allowed at* n_j **then**
            `// The plain way of computing` τ_j `is:`
            $\tau_j \leftarrow \min_{t \in [\lambda_j, T]}\{opt_j(t).val\}$;
            /* Note that heap optimization is used to speed up the computation */
          $e \leftarrow (n_j, n_i)$;
          for *each possible travel time* $w \in$ *[λ_i - T, θ_i - T]* **do**
            $q \leftarrow$ **Compute-Minimum-Fuel-Cost**(e, T, w, $v_{max}(e,t)$, t_k, t_m);
            $opt_{j\to i}(T+w) \leftarrow \min\{opt_{j\to i}(T+w), \tau_j + f_{j,i}(T) + q.c_{j,i}(T, q.\mathbf{v})\}$;
            if $opt_i(T+w) > opt_{j\to i}$ **then**
              `// if so, we use` $opt_{j\to i}$ `to update` $opt_i(T+w)$
              $opt_i(T+w).val \leftarrow opt_{j\to i}(T+w)$;
              $opt_i(T+w).pre \leftarrow n_j$;
              $opt_i(T+w).q \leftarrow q$;
              $opt_i(T+w).preCost \leftarrow \tau_j$;
$\tau_e \leftarrow \min_{t \in [\lambda_e, t_a]}\{opt_e(t).val\}$;
if $\tau_e = \infty$ **then**
  return "No feasible routes exist!";
else
  $t_e \leftarrow \min\{t | opt_e(t).val = \tau_e, t \in [\lambda_e, t_a]\}$;
  return $opt_e(t_e)$, τ_e, t_e;

Figure 5.4: Algorithm 4: COMPUTE-MINIMUM-COST (Step 4).

19), if waiting is disallowed at n_j, let τ_j denote the current $opt_j(T).val$, i.e., the minimum cost when arriving at node n_j exactly at time T (Lines 20 - 21); if waiting is allowed at n_j, then let τ_j denote the current minimum cost of $opt_j(t).val$ among all $t \in [\lambda_j, T]$, i.e., the minimum cost when arriving at node n_j before or at time T (Lines 22 - 25).

Heap optimization to obtain τ_j. In the case that waiting is allowed at n_j, the traditional way to obtain τ_j is to perform a loop to find the minimum value of $opt_j(t).val$ for all $t \in [\lambda_j, T]$ (Line 24), whose time complexity is $O(T - \lambda_j)$, and whose time complexity is $O(\theta_j - \lambda_j)$ in the worst case. In our implementation, in order to compute the value of τ_j more efficiently, we use a binary min-heap for each node n_j to maintain the values of $opt_j(t)$ for all $t \in [\lambda_j, T]$. The min-heap always maintains the specific $opt_j(t)$ with the minimum value of $opt_j(t).val$ among all $t \in [\lambda_j, T]$ as the root element. Thus finding the minimum takes only O(1) time. Each insertion and each deletion take $O(\log(\theta_j - \lambda_j))$ time at the node n_j. Therefore, compared to the plain way of finding the minimum, the min-heap structure helps reduce much time complexity.

Next, for edge $e = (n_j, n_i)$, since the domain of the arrival time at node n_i is $[\lambda_i, \theta_i]$, then the range of travel time of this edge e, i.e., w, should be $[\lambda_j - T, \theta_j - T]$. Hence, Lines 27 - 35 of Algorithm 4 (COMPUTE-MINIMUM-COST) perform a loop for each travel time stamp w. Line 28 calls Algorithm 1 (Compute-Minimum-Fuel-Cost) to obtain a quadruple which specifies the minimum fuel cost $q.c_{j,i}(T, q.\mathbf{v})$ when the travel time of edge e is exactly w. If the current $opt_{j\to i}(T+w)$ is larger than $\tau_j + f_{j,i}(T) + q.c_{j,i}(T, q.\mathbf{v})$, then Line 29 uses $\tau_j + f_{j,i}(T) + q.c_{j,i}(T, q.\mathbf{v})$ to update $opt_{j\to i}(T + w)$. Finally (Lines 30 - 35), we use $opt_{j\to i}(T + w)$ to update $opt_i(T + w)$. Let $t = T + w$, then $opt_i(T + w)$ is rewritten as $opt_i(t)$. Note that the current $opt_i(t)$ is not guaranteed to be the final optimal (or correct) OC-function. In conclusion, Algorithm 4 updates $opt_i(t)$ iteratively for node n_i by three loops: the first-layer loop (Lines 16 - 35) iterates each incoming neighbor n_j of n_i; the second-layer loop (Lines 19 - 35) iterates each departure time T from n_j; the third-layer (Lines 27 - 35) loop iterates each $w \in [\lambda_i - T, \theta_i - T]$. In this way, $opt_i(t).val$ approaches its optimal value.

In the last iteration of the outermost loop, Algorithm 4 processes the last node of *Vec*,

i.e., the destination node n_e. During this iteration, Algorithm 4 processes the incoming edges of n_e and updates $opt_e(t)$ iteratively.

Ultimately, Lines 39 - 41 of Algorithm 4 find $opt_e(t_e)$ and corresponding time stamp t_e.

5.2.5 Backtracking the Cost-Optimal Route

Algorithm 5: BACKTRACK-OPTIMAL-ROUTE(G_T,$opt_e(t_e)$,t_e)

Input: G_T,$opt_e(t_e)$,t_e
Output: Cost-optimal route R

1 $R\leftarrow$ an empty vector of sextuples <*node*, t_{arr}, γ, q, *minCost*, $next\_node$>;
2 *ReverseR*$\leftarrow$ an empty vector of quintuples <n_i, t_i, γ_i, q, opt_i>;
3 $n_i \leftarrow n_e$;
4 $t_i \leftarrow t_e$;
5 $\gamma_i \leftarrow 0$;
6 $q \leftarrow$ *NULL*;
7 $opt_i \leftarrow opt_e(t_e)$;
8 **while** $n_i \neq n_s$ **do**
9 *ReverseR*.push_back(<n_i, t_i, γ_i, q, opt_i>);
10 $n_j \leftarrow opt_i.pre$;
11 $t_j \leftarrow \min\{t|opt_j(t).val = opt_i.preCost, t \in [\lambda_j, opt_i.q.T]\}$;
12 $\gamma_j \leftarrow opt_i.q.T - t_j$;
13 $q \leftarrow opt_i.q$;
14 $opt_j \leftarrow opt_j(t_j)$;
15 $i \leftarrow j$;
16 **if** $n_i = n_s$ **then**
17 *ReverseR*.push_back(<n_i, t_i, γ_i, q, opt_i>);
18 ***break***;
19 **for** *id = ReverseR.size() − 1 to 0* **do**
20 *node* $\leftarrow$ *ReverseR*[*id*].n_i;
21 $t_{arr} \leftarrow$ *ReverseR*[*id*].t_i;
22 $\gamma \leftarrow$ *ReverseR*[*id*].γ_i;
23 $q \leftarrow$ *ReverseR*[*id*].q;
24 *minCost*$\leftarrow$ *ReverseR*[*id*].$opt_i.val$;
25 **if** *id equals 0* **then**
26 *next_node* $\leftarrow$ *NULL*;
27 **else**
28 *next_node* $\leftarrow$ *ReverseR*[$id - 1$].n_i;
29 R.push_back(<*node*,t_{arr},γ,q,*minCost*,*next_node*>);
30 **return** R;

Figure 5.5: Algorithm 5: BACKTRACK-OPTIMAL-ROUTE (Step 5).

We now present the algorithm for the last step of the ALG-COTER, i.e., backtracking the cost-optimal route R from destination n_e to source n_s and computing the waiting time $\gamma(n_i)$ for every node $n_i \in R$ such that $\mathsf{cost}(R) = opt_e(t_e).val$.

We first introduce the data structure. The reversed optimal route is stored in a vector named "*ReverseR*". Each element of this vector encapsulates: (i) the current node n_i, (ii) the optimal arrival time t_i at n_i, (iii) the waiting time γ_i, (iv) a quadruple q and (v) a quadruple opt_i. The quadruple q specifies a user leaves the previous node at departure time $q.T$ with speeds $\mathbf{v}$ and arrives at n_i at time instance t_i. The quadruple opt_i equals to $opt_i(t_i)$ specifying the minimum cost $opt_i.val$ when arriving at n_i at time t_i, the previous node $opt_i.pre$, $opt_i.q$, and $opt_i.preCost$, as discussed in the second paragraph in Section 5.2.4.1.

Then we use R to store the optimal route from source n_s to destination n_e by traversing each element in $ReverseR$ backward. Each element of R encapsulates (i) the current node $node$; (ii) the arrival time $\mathsf{Arr}(node) = t_{arr}$ at node $node$; (iii) the waiting time $\gamma(node)$ at node $node$, (iv) a quadruple q which specifies how a user departs from $node$ to the next node, i.e., at which departure time $q.T$, at which speeds $q.\mathbf{v}$, etc; (v) the current minimum cost $minCost$ when arriving at $node$ at t_{arr}; (vi) the successor node $next\_node$. Observe R contains very detailed information about the cost-optimal route. Therefore it is guaranteed that a user following route R step by step spends the minimum cost (fuel cost and toll fee).

The algorithm is denoted as BACKTRACK-OPTIMAL-ROUTE (Algorithm 5) and depicted in Fig. 5.5. The key idea of the BACKTRACK-OPTIMAL-ROUTE algorithm is to find the predecessor iteratively for every node of the optimal route backward from destination n_e to source n_s. Initially (Lines 3 - 7), $n_i \leftarrow n_e$. We initialize $opt_i(t_i)$ as $opt_e(t_e)$ and t_i as t_e. In every iteration, Lines 10 - 15 are used to find the predecessor n_j of n_i in the optimal route R. Thereby $n_j = opt_i(t_i).pre$ is the predecessor of n_i in the optimal route R, $opt_i(t_i).q.T$ is the departure time from n_j, $opt_i(t_i).q.\mathbf{v}$ is the speed with which the user travels on edge (n_j, n_i). $t_j = \min\{t | opt_j(t_j).val = opt_i(t_i).preCost,\ t \in [\lambda_j, opt_i(t_i).q.T]\}$ is the arrival time at n_j in the optimal route R, and $opt_j(t_j).val =$

$opt_i(t_i).preCost$. Such a predecessor n_j is guaranteed to exist, since $opt_i(t_i).val$ is computed by Algorithm 4 using $opt_j(t_j).val$ as expressed in Eq. 5.12. The waiting time at $n_j \in R$ is given as Eq. 5.13.

$$\gamma(n_j) = opt_i(t_i).q.T - t_j \tag{5.13}$$

The 'while-loop'(Lines 8 - 18) in Algorithm 5 finds the predecessor node one by one from the destination n_e, and terminates when the source n_s is found as a predecessor (Lines 16 - 18), i.e., $n_i = n_s$. Here, all the nodes in the optimal route are found and the waiting time $\gamma(n_i)$ for $n_i \in R$ is computed.

The 'for-loop' (Lines 19 - 29) in Algorithm 5 traverses each element of $ReverseR$ backward. For each iteration, Lines 20 - 28 extract the current node $node$, the optimal arrival time t_{arr} at node $node$, the waiting time γ at $node$, the quadruple q which specifies how a user departs from $node$ to the next node, and the next node $next\_node$. Line 29 stores each node of the optimal route from source n_s to destination n_e with related information into the vector R. The first element in R is a sextuple containing n_s, t_{arr}, γ, q, $minCost$, $next\_node$, and corresponds to the last element in $ReverseR$. The 'for-loop' terminates when the sextuple which contains n_e is pushed into R. Note that for node n_e (Lines 25 - 26), the successor node $next\_node$ is set as *NULL*.

5.2.6 Time Complexity Analysis

We analyze the time complexity of our ALG-COTER algorithm (Algorithm 2) in this subsection.

For Step 1 and Step 2, the single-source shortest path algorithm, namely, Fibonacci-heap optimized Dijkstra, is used. Its time complexity in the worst case is $O(|V| \log |V| + |E|)$.

For Step 3, i.e., TOPOLOGICAL-SORT algorithm (Algorithm 3), as analyzed in Section 5.2.3, the time complexity in the worst case is $O(|V| + |E|)$.

For Step 4, i.e., computing the minimum cost $opt_i(t)$ for node n_i. Algorithm 4 COMPUTE-MINIMUM-COST (Fig. 5.4) conducts a loop (Lines 6 - 35) to process each

node in the topologically sorted vector *Vec*. The maximum size of *Vec* (in the worst case) is $|V|$. (In fact, from our experiments, we find the size of *Vec* is usually much smaller than $|V|$, i.e., $|Vec| << |V|$.) In the *it*-th iteration of the first-layer loop, the current node n_i is $Vec[it]$, and Algorithm 4 performs a second-layer loop (Lines 16 - 35) to process n_i's incoming edges. It is easy to know that the number of $Vec[it]$'s incoming edges is at most it, that is, the number of nodes which locate before $Vec[it]$ in *Vec*. Thus the total time complexity for the first-layer loop and the second-layer loop is $\sum_{it=0}^{it=|Vec|-1} it = \frac{|Vec|^2-|Vec|}{2}$. (In the worst case, the total number of operations of the first-layer loop and the second-layer loop is $O(|V| + |E|)$, which means in the worst case, the whole road network is involved in the topological sorting.) For each incoming neighbor n_j, the departure time T from n_j may have at most $(\theta_j - \lambda_j)$ possible time stamps. For every time stamp T, Algorithm 4 finds τ_j by heap optimization with time complexity $O(\log(\theta_j - \lambda_j))$, and then performs another loop (Lines 27 - 35) for each possible travel time $w \in [\lambda_i - T, \theta_i - T]$. For each w, Algorithm 4 calls Algorithm 1 (Compute-Minimum-Fuel-Cost) once. The time complexity of Algorithm 1 is $O((m-k+1)^2)$ since we can simply use genetic algorithm to solve the nonlinear optimization problem. Therefore, the time complexity of Algorithm 4 is:

$$
\begin{aligned}
& T(\text{Step } 4) \\
&= O(\min\{\frac{|Vec|^2 - |Vec|}{2}, |V| + |E|\}) \cdot \max_{n_j \in \text{Vec}} \{\theta_j - \lambda_j\} \\
&\quad \cdot (O(\log(\theta_j - \lambda_j)) + (O(\theta_i - T - (\lambda_i - T)) \cdot O((m-k+1)^2))) \\
&\leq O(\min\{\frac{|Vec|^2 - |Vec|}{2}, |V| + |E|\}) \\
&\quad \cdot (t_a - t_d) \cdot (O(\log(t_a - t_d)) + O(t_a - t_d) \cdot O(p^2)) \\
&= O((t_a - t_d)^2 p^2 \cdot \min\{\frac{|Vec|^2 - |Vec|}{2}, |V| + |E|\})
\end{aligned}
$$

For Step 5, i.e., backtracking the optimal route R and computing the waiting time for each node in R, which is shown in Fig. 5.5. The time complexity of the 'while-loop' and the time complexity of the 'for-loop' are $O(|Vec|)$, and in the worst case, $O(|Vec|) = O(|V|)$.

In conclusion, the total time complexity of our ALG-COTER algorithm in the worst

case is given as follows:

$$
\begin{aligned}
& T(\text{ALG-COTER}) \\
&= O(|V|\log|V| + |E|) + O(|V| + |E|) \\
&+ O((t_a - t_d)^2 p^2 \cdot (\min\{\frac{|\mathit{Vec}|^2 - |\mathit{Vec}|}{2}, |V| + |E|\}))) + O(|V|) \\
&= \max\{O(|V|\log|V| + |E|), O((t_a - t_d)^2 p^2 \min\{\frac{|\mathit{Vec}|^2 - |\mathit{Vec}|}{2}, |V| + |E|\})\}
\end{aligned}
$$

Discussion: The above equation gives the upperbound of the time complexity of our algorithm in the worst case. Actually, the number of the candidate nodes $|\mathit{Vec}|$ is much smaller than $|V|$, usually no more than 100; and the degree of each node in road networks is very small, no more than 10. Therefore, $\frac{|\mathit{Vec}|^2 - |\mathit{Vec}|}{2} < 10^3 << |V| + |E|$.

5.3 Experiments

In this section, we study the performance of our proposed ALG-COTER algorithm.

5.3.1 Experimental Datasets

We use three real-world road networks: City of Oldenburg (OL), City of San Joaquin County (TG) Road Network[†], and Florida (FLA) Road Network[‡]. OL has 6,105 nodes and 7,035 edges. TG has 18,263 nodes and 23,874 edges. FLA has 1,070,376 nodes and 2,712,798 edges. The average lengths of edges in OL, TG, and FLA road network are 73.679, 34.9, and 0.2043 km, respectively. Note that each road segment in the original FLA corresponds to two edges which have opposite directions. we only preserve the unidirectional edges and delete the redundant opposite edges to avoid loops. Thus the number of valid edges in FLA is halved to 1,456,400, as given in Table 5.2. Meanwhile we get a directed acycline graph of FLA, which satisfies the requirement of topological sorting.

[†]http://www.cs.utah.edu/%7Elifeifei/SpatialDataset.htm
[‡]http://www.dis.uniroma1.it/challenge9/download.shtml

5.3.2 Simplified Toll Fee Function for Experiments

Note that our algorithm can deal with any toll fee function with respect to the departure time as long as the toll fee function is an ***arbitrary single-valued function***. For the sake of convenience, without loss of generality, we temporarily use the following toll fee function in our experiments:

$$f_{i,j}(T) = \begin{cases} f_1; & T_0 \leq T \leq T_1 \\ f_2; & T_1 < T \leq T_2 \\ \cdots & \\ f_l; & T_{l-1} < T \leq T_l \end{cases} \tag{5.14}$$

where, $[T_0, T_l]$ is the time domain of function $f_{i,j}(T)$. f_x $(1 \leq x \leq l)$ is a constant value, which represents the value of $f_{i,j}(T)$ when $T \in (T_{l-1}, T_l]$. Specifically, $T_0 = t_d$, $T_l = t_a$.

5.3.3 Experimental Objective and Perspective

We evaluate the efficiency, sensitivity, and scalability, of our ALG-COTER algorithm from the following aspects: (i) running time with respect to the number of nodes; (ii) running time with respect to the number of candidate nodes and the number of candidate routes; (iii) running time with respect to the distances between the source n_s and destination n_e; (iv) running time with respect to the length of time interval $[t_d, t_a]$; (v) running time with respect to the average number of piecewise intervals of $f_{i,j}(T)$; (vi) running time with respect to the number of piecewise intervals of $v_{max}(e, t)$; (vii) running time with respect to the average length of edges; (viii) running time of ALG-COTER vs. running time of a baseline method.

Table 5.2: Input road networks.

graph	♯nodes	♯ valid edges	average length of edges
OL	6,105	7,035	73.679 (km)
TG	18,263	23,874	34.9 (km)
FLA	1,070,376	1,456,400	0.2034 (km)

Experiments are implemented in C/C++ Microsoft Visual Studio 2010 on a machine

with an Intel(R) Xeon(R) CPU X5650 2.67GHz, 6 cores and 24.0 GB of memory running Windows 7 professional.

5.3.4 Default Values of Parameters and Experimental Setup

Table 5.3 lists the default values of some parameters. All the experiments below are conducted by changing the value of one parameter while keeping the default values of other parameters.

Table 5.3: Default values of some parameters.

parameter	**default value**
t_d	0
t_a	1440 min
l: number of segments of $f_{i,j}(T)$ for $T \in [t_d, t_a]$	100
p: number of segments of $v_{max}(e,t)$ for $t \in [t_d, t_a]$	24
V_{max}	130 (km/h)
V_{min}	40 (km/h)
number of candidate routes from n_s to n_e	4 (OL & TG); 3 (FLA)
number of candidate nodes in *Vec* for OL	$9 \sim 10$
number of candidate nodes in *Vec* for TG	$8 \sim 10$
number of candidate nodes in *Vec* for FLA	17
avglen of OL, TG, FLA	73.68, 34.9, 0.2034 (km)
distance from n_s to n_e in OL, TG, FLA	600, 75, 3 (km)

Unless specified in particular, we conduct experiments on the original OL or TG or FLA network, which means the number of nodes, number of edges and avglen of OL, TG and FLA are the same as those in Table 5.2. In our experiments, t_d is always set as 0, and t_a is set as 1440 by default. Note that we use minutes to measure the time stamps.

Notice that the default value of p, i.e., number of segments of v_{max} (e, t), is 24. As mentioned in Definition 16, $V_{max}(e,t)$ = 130 km/h and $V_{min}(e,t)$ = 40 km/h. Hence we assume the values of $v_{max}(e, t)$ are from the array $v[10]$ = {130, 120, 110, 100, 90, 80, 70, 60, 50, 40}. We divide the whole time domain into p = 24 equal intervals, and the length of each interval is $\frac{t_a - t_d}{p}$ minutes. For each time interval $[t_k, t_{k+1}]$ where $k \in [0, p-1]$, the maximum speed allowed on edge e during this interval, i.e., $v_{max}(e,t)$, is

set as $v[(e.ID + k)\%10]$, where $e.ID$ is an integer denoting the identifier of edge e. In this way, we guarantee that the number of segments of $v_{max}(e, t)$ for each edge e is a fixed number, equal to p.

Without loss of generality, likewise, we set the values of $f_{i,j}(T)$ for $T \in [t_d, t_a]$ for each edge $e = (n_i, n_j)$ by Eq. 5.14. Suppose the number of segments of $f_{i,j}(T)$ for edge e, i.e., l, is 100, then the whole time domain $[t_d, t_a]$ is divided into 100 equal time intervals. We use an array $f[4] = \{20, 15, 10, 5\}$ to simulate the values of the toll fee for each edge. For $T \in [T_{x-1}, T_x]$ where $x \in [1, l]$, the value of $f_{i,j}(T)$ is set to be $f[(e.ID + (x-1))\%4]$, where $e.ID$ is the identifier of edge e. In this way, we guarantee the number of segments of $f_{i,j}(T)$ for each edge is a fixed value, equal to l.

Note that we keep the default value of the number of candidate routes as 4 for OL and TG road networks, and the default value of the number of candidate routes for FLA road network is 3. Likewise, we keep the default value of the number of candidate nodes in *Vec* for the OL, TG, FLA networks as 9$\sim$10, 8$\sim$10, 17, respectively. In fact, in order to study the influence of the number of candidate routes on the running time, we compute all the routes from n_s = 0 to all nodes reachable by it for the OL network, and compute all the routes from n_s = 23 to all nodes reachable by it for the TG network, and we also compute all the routes from n_s = 83 to all nodes reachable by it for FLA network, regardless of the time constraint, in our preprocessing step. Thus, we have known there are how many routes from n_s = 0 to any node in the OL network, how many routes from n_s = 23 to any node in the TG network, and how many routes from n_s = 83 to any node in the FLA network in advance. Thereby, we can control the number of candidate routes in our experiments.

5.3.5 Experimental Results

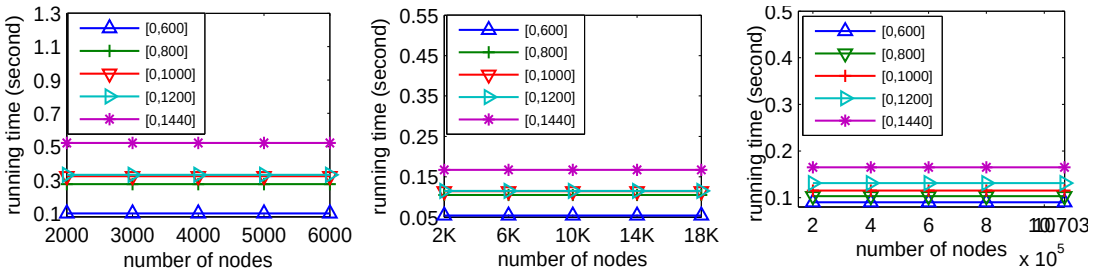
We now report the detailed experimental results.

5.3.5.1 Exploring the influence of the number of nodes

We first generate smaller subgraphs from the three original road networks. The number of nodes in the subgraphs of OL increases from 2K to 6K. The number of nodes in the subgraphs of TG increases from 2K to 18K. The number of nodes in the subgraphs of FLA increases from 0.2 million to 1.07 million. The process of generating smaller graphs is as follows. Suppose the number of nodes in the smaller graph is N_i. From the original nodes set V, we preserve the first N_i nodes and abandon the remaining nodes. From the original edge set E, we only preserve the edges whose starting node and ending node are among the first N_i nodes. Thus, a subgraph is generated. The details of generated subgraphs of OL, TG and FLA are given in Table 5.4, 5.5, and 5.6, respectively.

The number of piecewise intervals of $f_{i,j}(T)$ and number of piecewise intervals of $v_{max}((n_i,n_j),t)$ are set as their default values 100, and 24. To better reflect the relation between the runtime of our ALG-COTER and the number of nodes of a graph, we conduct 5 groups of experiments: the entire time interval [t_d, t_a] is set as [0, 600], [0, 800], [0, 1000], [0, 1200], and [0, 1440], respectively. In each group, as said in the last paragraph of Section 5.3.4, we always choose suitable n_e to guarantee the number of candidate nodes in *Vec* and the number of candidate routes from n_s to n_e are their default values in Table 5.3, to avoid the effects of the number of candidate nodes or routes on runtime.

In Fig. 5.6, we explore the influence of the number of nodes on the running time on the OL, TG, and FLA road network, respectively. As shown in Fig. 5.6(a), 5.6(b), and 5.6(c), the running time of ALG-COTER is hardly affected by the changes of the total number of nodes, which means the running time has a low sensitivity to the number of nodes in a graph. The reason is that the difference of the number of candidate nodes is only 1 or 2, and the number of candidate routes is kept the same, which means the search space of our ALG-COTER algorithm does not change much and guarantees the increases of nodes and edges do not contribute to the search space of the subsequent algorithms (Algorithms 4 and 5). Meanwhile, we can also find our algorithm is scalable to the number of nodes N_i.



(a) OL road network (b) TG road network (c) FLA road network

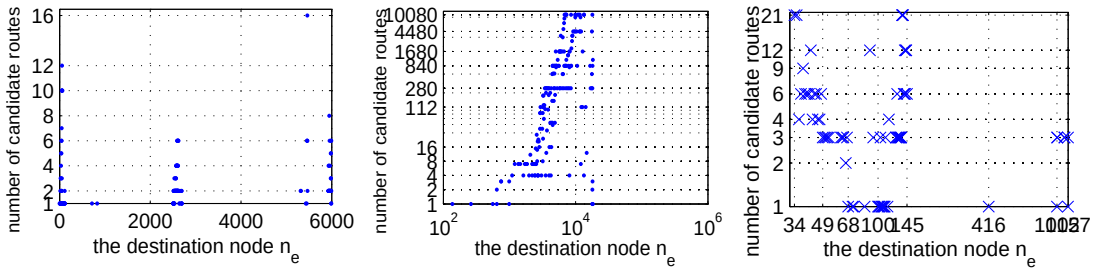
Figure 5.6: The runtime with respect to the number of nodes.

5.3.5.2 Exploring the influence of the number of candidate nodes in *Vec* and candidate routes

We carry out experiments to study the influences of the number of candidate nodes in *Vec* and candidate routes from the source n_s to the destination n_e on running time. From Section 5.3.4, if we set n_s as a fixed node, then we can group the nodes n_i that are reachable by n_s according to how many routes existing from n_s to n_i in our preprocessing step. Meanwhile, if we set n_s as a fixed node, then we can also group the nodes by how many candidate nodes in *Vec* from n_s to n_i.

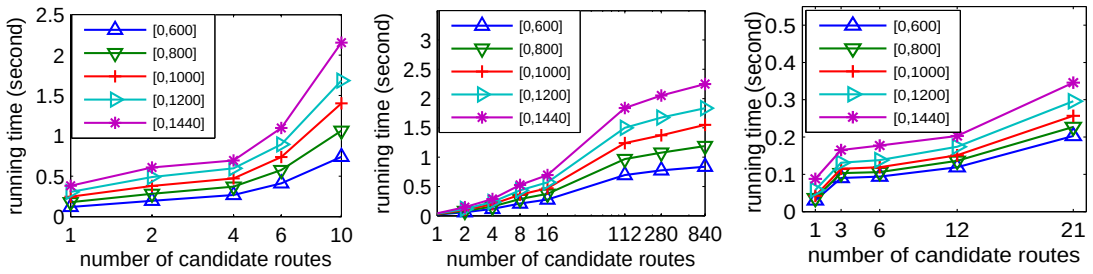
For the OL road network, we set n_s = 0, and we find that the number of routes from n_s to any other node reachable by n_s ranges between 1 to 16. We plot the number of candidate routes as a function of the identifier of n_e in Fig. 5.7(a). The numbers of candidate routes which have high frequency are 1, 2, 3, 4, 5, and 6. However, the numbers such as 7, 8, 10, 12, 16 also exist although they have relatively low frequency. Therefore, in order to study the effects of the number of candidate routes, we depict the average running time as a function of the number of candidate routes in Fig. 5.8(a). Meanwhile, the numbers of candidate nodes in *Vec* which have high frequency are 6, 7, 9, 13, and 26, respectively. Hence, we also depict the average running time as a function of the number of candidate nodes in *Vec*, as shown in Fig. 5.9(a).

For the TG road network, we set n_s = 23, and we find many nodes are reachable by n_s and a great many routes exist from n_s to its reachable nodes, as shown in Fig. 5.7(b). The numbers of routes which have high frequency are 1, 2, 3, 4, 6, 8, 16, 112, 280, 840, 1680, 4480 and 10080. In fact, even if the number of routes is large, the number of candidate



(a) OL, $n_s = 0$ (b) TG, $n_s = 23$ (c) FLA, $n_s = 83$

Figure 5.7: The number of candidate routes with respect to n_e from fixed n_s.

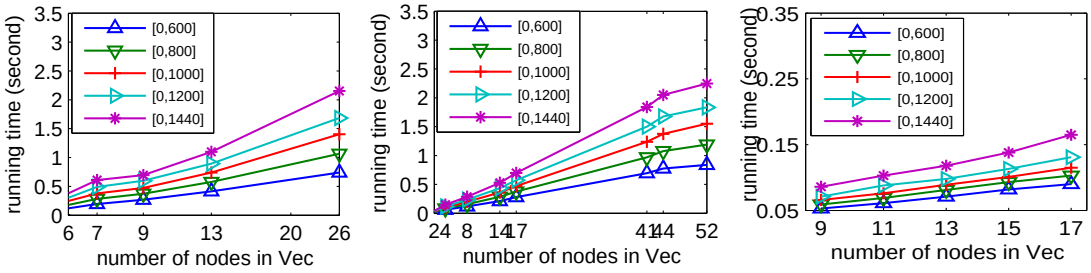


(a) OL road network (b) TG road network (c) FLA road network

Figure 5.8: The runtime with respect to the number of candidate routes.

nodes in *Vec* is still small. For instance, from n_s = 23 to n_e = 6805, there are 10080 different routes but the number of candidate nodes is only 76. Imagine that for Algorithm 4, if we perform iterations for each candidate route instead of performing iterations for candidate nodes, the time complexity will be definitely much higher. This fact strongly demonstrates that the topological sorting in Step 3 can reduce the search space of Step 4. In order to explore the effects of the number of candidate routes, we depict the average running time as a function of the number of candidate routes in Fig. 5.8(b). Meanwhile, the numbers of candidate nodes in *Vec* which have high frequency are 2, 4, 8, 14, 17, 41, 44 and 52, respectively. Hence, we also depict the average running time as a function of the number of candidate nodes in *Vec*, as shown in Fig. 5.9(b).

For the FLA road network, we set n_s = 83, and we find the number of routes from n_s to other nodes ranges between 1 to 21. We plot the number of candidate routes as a function of n_e in Fig. 5.7(c). The numbers of candidate routes which have high frequency are 1, 3, 6, 12 and 21. We depict the average running time as a function of the number of candidate routes in Fig. 5.8(c). We also depict the average running time as a function of the number of candidate nodes in *Vec* while keeping the number of routes unchanged, as



(a) OL road network (b) TG road network (c) FLA road network

Figure 5.9: The runtime with respect to the number of candidate nodes.

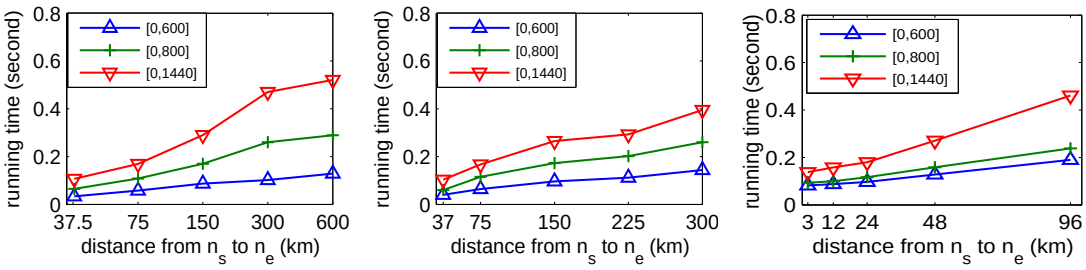
shown in Fig. 5.9(c).

In Fig. 5.8 and 5.9, we conduct 5 groups of experiments: t_a is set as 600, 800, 1000, 1200, and 1440, respectively. Observe the running time increases when the number of candidate routes or the number of candidate nodes increases in each group of experiment.

First, it is straightforward that if more candidate routes exist, which means each node may have more incoming edges, then Algorithm 4 (COMPUTE-MINIMUM-COST) must perform more iterations for the second-layer loop (Lines 12 - 35). Second, if there exist more candidate routes from n_s to n_e, more nodes may be involved, which means the size of *Vec* may be larger and thus Algorithm 4 (COMPUTE-MINIMUM-COST) must perform more iterations for the outermost loop (Lines 6 - 35). This is why the average runtime increases when the number of candidate routes increases in each group of experiment shown in Fig. 5.8.

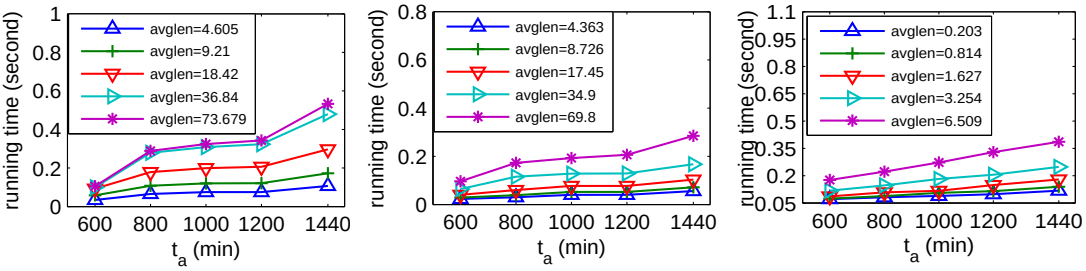
Likewise, from Algorithm 3 (TOPOLOGICAL-SORT) and Algorithm 4 (COMPUTE-MINIMUM-COST), we know that if more candidate nodes exist in the topologically sorted vector *Vec*, which means there are more nodes involved from n_s to n_e, then the running time should be relatively larger, since Algorithm 3 (TOPOLOGICAL-SORT) may have larger depth and Algorithm 4 (COMPUTE-MINIMUM-COST) may have to perform more iterations. This is why the average running time increases when the number of candidate nodes in *Vec* increases in each group of experiment shown in Fig. 5.9.

From Fig. 5.8 and 5.9, we can also find that our algorithm is efficient even when the number of routes or number of nodes in *Vec* is very large. In addition, we also see our algorithm is scalable to the number of candidate routes.



(a) OL road network (b) TG road network (c) FLA road network

Figure 5.10: The runtime with respect to the distances between n_s and n_e.



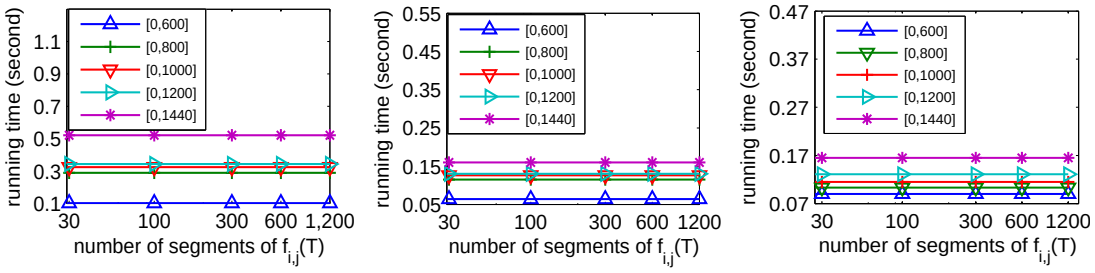
(a) OL road network (b) TG road network (c) FLA road network

Figure 5.11: The runtime with respect to the length of time interval $[t_d, t_a]$.

5.3.5.3 Exploring the influence of the distances between n_s and n_e

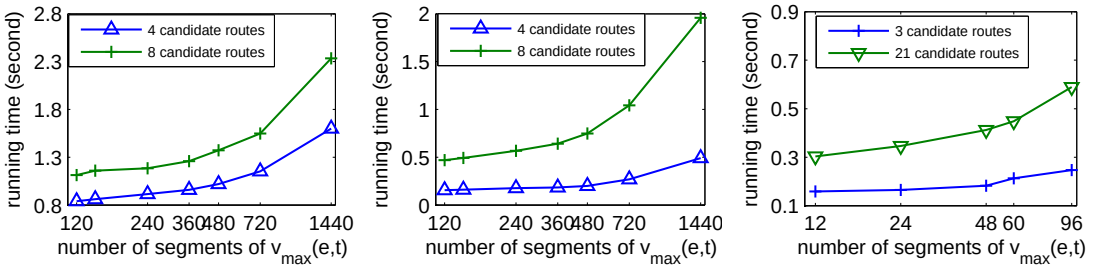
We conduct a group of experiments to study the influence of the distances between n_s and n_e on runtime. Usually, the number of edges in a route from n_s to n_e may be affected when the distances between n_s and n_e increase.

If the number of edges from n_s to n_e increases, then (i) the recursive depth of Algorithm 3 (TOPOLOGICAL-SORT) is larger and thus Algorithm 3 search more edges to traverse n_e until n_s; (ii) Algorithm 4 (COMPUTE-MINIMUM-COST) computes $opt_i(t)$ iteratively for more edges; and (iii) the optimal route may contain more nodes and thus Algorithm 5 (BACKTRACK-OPTIMAL-ROUTE) performs more iterations to get the whole optimal route.



(a) OL road network (b) TG road network (c) FLA road network

Figure 5.12: The runtime with respect to the average number of piecewise intervals of $f_{i,j}(T)$.



(a) OL road network (b) TG road network (c) FLA road network

Figure 5.13: The runtime with respect to the number of piecewise intervals of $v_{max}(e,t)$.

As analyzed above, the runtime should increase if the distance between n_s and n_e increases. In Fig. 5.10(a), 5.10(b), 5.10(c), the distance from n_s to n_e ranges from 37 to 600 km, 37 to 300 km, 3 to 96 km, for OL, TG, and FLA. Observe that the running time is still small even when the distance between n_s to n_e is very large, which proves the efficiency of our algorithm. Meanwhile, observe that the running time increases when the distance between n_s and n_e increases on all road networks, which indicates our algorithm is scalable with respect to the distance from n_s to n_e.

5.3.5.4 Exploring the influence of the length of time interval $[t_d, t_a]$

We conduct experiments to study the influences of the length of time interval $[t_d, t_a]$ on the running time. The earliest departure time t_d is fixed as 0, while the latest arrival time t_a ranges from 600 to 1440. avglen is the average length of edges of a road network. From Table 5.2, we know avglen is originally 73.679, 34.9 and 0.2034 for OL, TG and FLA.

In Fig. 5.11(a), we keep the number of nodes and number of edges the same, and

set the length $len(e)$ of each edge e to $len(e)$/16, $len(e)$/8, $len(e)$/4, $len(e)$/2 and $len(e)$ respectively. Thus avglen of OL becomes $\frac{73.679}{16}$ = 4.605, $\frac{73.679}{8}$ = 9.21, $\frac{73.679}{4}$ = 18.42, $\frac{73.679}{2}$ = 36.84, and 73.679, respectively.

In Fig. 5.11(b), we set the length $len(e)$ of each edge e to $len(e)$/8, $len(e)$/4, $len(e)$/2, $len(e)$ and $len(e)$*2, respectively. Thus avglen of TG becomes $\frac{34.9}{8}$ = 4.363, $\frac{34.9}{4}$ = 8.726, $\frac{34.9}{2}$ = 17.45, 34.9, and 34.9*2 = 69.8, respectively.

In Fig. 5.11(c), we set the length $len(e)$ of each edge e to $len(e)$, $len(e)$*4, $len(e)$*8, $len(e)$*16 and $len(e)$*32, respectively. Thus avglen of FLA becomes 0.2034, 0.2034*4 = 0.8136, 0.2034*8 = 1.6272, 0.2034*16 = 3.2544, and 0.2034*32 = 6.5088, respectively.

From Fig. 5.11, we see the running time is still small when the time interval is large, which demonstrates our algorithm is efficient. Observe that the running time increases with the increase of the length of the time intervals, for different average lengths of edges. The reason is Algorithm 4 (COMPUTE-MINIMUM-COST) performs a loop (Lines 19 - 35) for each departure time instance $T \in [\lambda_i, \theta_i]$. As the time interval $[t_d, t_a]$ becomes wider, the interval $[\lambda_i, \theta_i]$ becomes wider accordingly, since a user has looser time constraint and if t_a becomes larger, then θ_i becomes larger accordingly. Hence the loop (Lines 19 - 35) contains more iterations, and thus, more running time is required. Moreover, Fig. 5.11 indicates our algorithm is scalable to the length of the entire time interval.

5.3.5.5 Exploring the influence of the average number of piecewise intervals of $f_{i,j}(T)$

Recall we take a simplified model of the toll fee function in Eq. 5.14 for example in our experiments. We study the influence of the number of piecewise intervals of $f_{i,j}(T)$ on runtime. Let l represent the number of segments of $f_{i,j}(T)$ for $T \in [t_d, t_a]$ in Eq. 5.14. We vary l according to a set {30, 100, 300, 600, 1200} while the entire time interval $[t_d, t_a]$ does not change. The lengths of the piecewise intervals of $f_{i,j}(T)$ are set as follows: if $(t_a - t_d)\%l = 0$, then the length of each piecewise interval is $\frac{t_a - t_d}{l}$; otherwise, we set the length of each of the first $\lfloor l/2 \rfloor$ piecewise intervals as $\lfloor \frac{t_a - t_d}{l} \rfloor$, and set the length of each of the rest $\lfloor \frac{t_a - t_d - \lfloor \frac{t_a - t_d}{l} \rfloor \cdot \lfloor l/2 \rfloor}{\lceil \frac{t_a - t_d}{l} \rceil} \rfloor$ piecewise intervals as $\lceil \frac{t_a - t_d}{l} \rceil$; and the length of the

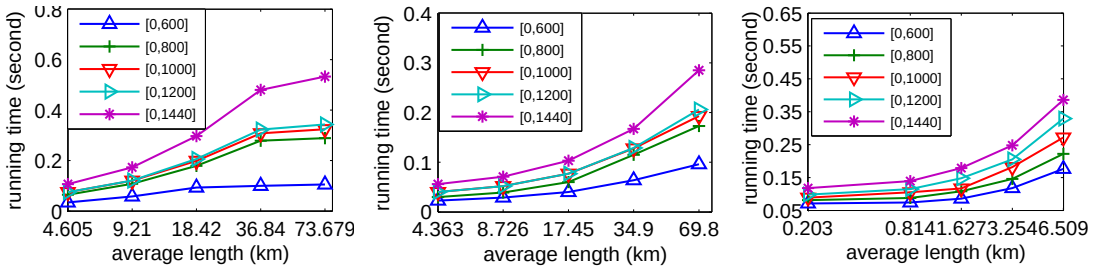
last interval is $(t_a - t_d - \lfloor \frac{t_a - t_d}{l} \rfloor \cdot \lfloor l/2 \rfloor) \% \lceil \frac{t_a - t_d}{l} \rceil$.

Figure 5.12 shows that the runtime is not affected by l. The reason is that Algorithm 4 (COMPUTE-MINIMUM-COST) processes each time instance T in the time interval [λ_i, θ_i] (Line 19), which is independent of the number of piecewise intervals of $f_{i,j}(T)$. In other words, our algorithm is not sensitive to the number of piecewise intervals of $f_{i,j}(T)$ and that's why our algorithm allows arbitrary toll fee functions.

5.3.5.6 Exploring the influence of the number of piecewise intervals of $v_{max}(e,t)$

We study the runtime with respect to the number of piecewise intervals of $v_{max}(e,t)$, i.e., p. The entire time interval [t_d, t_a] = [0, 1440] is kept unchanged while the value of p is set to 120, 144, 240, 360, 480, 720 and 1440, for OL or TG; and the value of p is set to 12, 24, 48, 60, and 96, for FLA. Thus the length of each piecewise interval of $v_{max}(e,t)$ is 12, 10, 6, 4, 3, 2 minutes and 1 minute for OL and TG; and the length of each interval is 120, 60, 30, 24, and 15 minutes for FLA. As the number of candidate routes affects the runtime, we use different groups of n_s and n_e (according to the number of candidate routes). The two groups shown in Fig. 5.13 contain 4 and 8 candidate routes for OL and TG, and contain 3 and 21 candidate routes for FLA.

From Fig. 5.13, we observe that the runtime increases with the increase of the number of segments of $v_{max}(e,t)$. This indicates that our algorithm is sensitive and scalable to the number of segments of $v_{max}(e,t)$. This is since when the number of segments of $v_{max}(e,t)$ is large, the travel time on edge e spans more piecewise intervals of $v_{max}(e,t)$, which means $(m-k+1)$ becomes larger. Under such circumstances, the dimensions of $\mathbf{v}$ and $\mathbf{t}$ get larger, and thus Algorithm 1 (Compute-Minimum-Fuel-Cost) shown in Fig. 5.1 has to take more steps for solving the nonlinear programming optimization problem (Line 1), which leads to higher runtime. In addition, observe that under the same conditions, running time on OL is always higher than that on TG. This is because the lengths of most edges in OL are longer than TG.



(a) OL road network (b) TG road network (c) FLA road network

Figure 5.14: The runtime with respect to the average length of edges.

5.3.5.7 Exploring the influence of the average length of edges

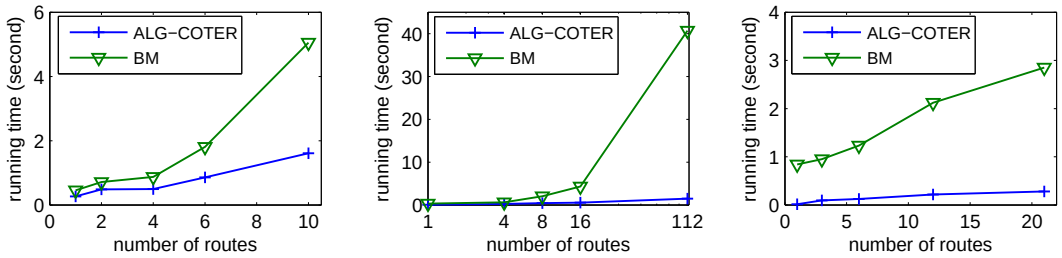
In Fig. 5.14, we depict the runtime as a function of the average length of edges.

For the OL road network, the number of nodes and number of edges are fixed as 6105 and 7035, but the length $len(e)$ of each edge e is set to $len(e)$/16, $len(e)$/8, $len(e)$/4, $len(e)$/2, $len(e)$, respectively. Hence the average length of edges in OL is 4.605, 9.21, 18.42, 36.84, 73.679 km, respectively.

For the TG road network, the number of nodes and the number of edges are fixed as 18,263 and 23,874, but the length of each edge e is set to $len(e)$/8, $len(e)$/4, $len(e)$/2, $len(e)$, $len(e)$*2, respectively. Hence the average length of edges in TG is 4.363, 8.726, 17.45, 34.9, 69.8 km, respectively.

For the FLA road network, the number of nodes and the number of edges are fixed as 1,070,376 and 1,456,400, but the length of each edge e is set to $len(e)$, $len(e)$*4, $len(e)$*8, $len(e)$*16 and $len(e)$*32, respectively. Thus avglen of FLA becomes 0.2034, 0.8136, 1.6272, 3.2544, 6.5088, respectively.

The number of piecewise intervals of $v_{max}((n_i, n_j), t)$ is 24. The entire time interval [t_d, t_a] ranges from [0, 600], to [0, 1440]. As shown in Fig. 5.14(a), 5.14(b), and 5.14(c), we find ALG-COTER runs fast with respect to different avglen on the three road networks. In addition, ALG-COTER runs faster if the average length is shorter, on the three road networks. This shows our algorithm is efficient, and is sensitive and scalable to the average length of edges.



(a) OL road network (b) TG road network (c) FLA road network

Figure 5.15: ALG-COTER vs. "BM".

5.3.5.8 Comparison with a baseline method

As our problem of COTER is novel, there are no baseline method for solving COTER. However, in order to show the superiority of our ALG-COTER algorithm, we design a baseline method which does not use topological sorting algorithm.

A baseline method. To distinguish from our ALG-COTER method, we name this baseline method as "BM". "BM" also has five steps and its first two steps and the last step are the same as that of our ALG-COTER. The differences between our ALG-COTER and "BM" lie in Step 3 and Step 4. For Step 3, "BM" computes all feasible candidate routes from n_s to n_e instead of computing the topological order of candidate nodes. For step 4, "BM" enumerates all candidate routes and in each iteration, it uses the recurrence formula to calculate the values of the optimal functions of the descendant nodes until the destination node n_e is reached.

We compare "BM" with our ALG-COTER algorithm. As shown in Fig. 5.15, we see that the runtime of "BM" is much higher than that of ALG-COTER. Therefore, we conclude that our ALG-COTER algorithm is more efficient than the baseline method "BM". These results strongly demonstrate the superiority and efficiency of our ALG-COTER algorithm.

5.4 Conclusion

We resolve the problem of the cost-optimal routing in time-dependent road networks (COTER) with time and speed constraints in this chapter. We allow waiting only at

some nodes, and at some other nodes, waiting is strictly forbidden. We consider two kinds of cost, i.e., fuel consumption cost and toll fee. We employ nonlinear programming optimization technique to compute the minimum fuel consumption on an edge when the travel time of this edge is fixed. We allow arbitrary single-valued toll fee functions with respect to different departure time for each edge. We propose an approximate ALG-COTER algorithm to solve COTER. Our ALG-COTER first computes the earliest arrival time λ_i for each node n_i and the latest arrival time θ_i for each candidate node n_i, by using the Fibonacci-heap optimized Dijkstra's algorithm. Subsequently, ALG-COTER conducts the topological sorting of all the candidate nodes that are reachable from n_s and meanwhile can reach n_e under time constraint. Afterwards, ALG-COTER uses dynamic programming, min-heap optimization, and nonlinear optimization techniques to compute the optimal cost functions $opt_i(t)$ at each feasible arrival time instance $t \in [\lambda_i, \theta_i]$ for each candidate node n_i iteratively, according to their topological sorting and the recurrence formula of their OC-functions, and finally obtains the minimum value of the optimal cost function at n_e. Ultimately, ALG-COTER traces back the cost-optimal route and computes the waiting time at each node in the optimal route. We also analyze the time complexity of our ALG-COTER algorithm. We evaluate the efficiency, sensitivity, and scalability of our ALG-COTER algorithm, by studying the influences of different parameters on running time. Experimental results on large-scale data sets demonstrate that our algorithm can find the minimum-cost route from a source to a destination under time and speed constraints efficiently, and is scalable to different parameters which have influences on the running time.

Table 5.1: Notations

Notation	Meaning
G_T	time-dependent road network G_T = (V, E, L, W, C, F), where V is the set of all nodes in G_T; E is the set of edges in G_T; L is the set of edge lengths; W is the set of the travel time for traversing each edge; C is the set of the fuel consumption for traversing each edge; F is the set of the toll fee cost for traversing each edge.
V_w; V_{nw}	the set of nodes which allow waiting; the set of nodes which disallow waiting
n_s; n_e	the source node; the destination node
t_d; t_a	the earliest departure time from n_s; the latest arrival time at n_e
$w_{i,j}(T, \mathbf{v})$	the travel time for traversing edge (n_i, n_j) with departure time T and speed $\mathbf{v}$ for $t \in [T, T + w_{i,j}(T, \mathbf{v})]$
$c_{i,j}(T, \mathbf{v})$	the fuel cost for traversing edge (n_i, n_j) with departure time T, and speed $\mathbf{v}$ for $t \in [T, T + w_{i,j}(T, \mathbf{v})]$
$v_{max}(e, t)$	the maximum velocity allowed on edge e at time t
p	the number of segments of $v_{max}(e, t)$
l	the number of segments of $f_{i,j}(T)$
v_i	the maximum velocity allowed during time interval (I_i, I_{i+1}] for $i \in [0, p-1]$
$\mathbf{t}$	the column vector of travel time on one edge as given in Eq. 5.6
$\mathbf{T}$	the column vector of departure time on one edge as given in Eq. 5.7
$\mathbf{v}$	the row vector of speed on one edge as expressed in Eq. 5.8
V_{max}	the upperbound of the maximum velocities allowed on each edge: 130 $\mathrm{km/h}$
V_{min}	the lowerbound of the maximum velocities allowed on each edge: 40 $\mathrm{km/h}$
avglen	the average length of the edges in G_T
$\gamma(n_i)$	the waiting time at node n_i
λ_i; θ_i	the earliest arrival time at node n_i; the latest arrival time at node n_i
$N^-(n_i)$; $N^+(n_i)$	the set of n_i's incoming neighbors; the set of n_i's outgoing neighbors
$opt_{j \to i}(t)$	the optimal (minimum) cost if arriving at n_i at t through edge (n_j,n_i)
$opt_i(t)$	the optimal-cost-function of n_i (a quadruple $\langle val, pre, q, preCost \rangle$)
t_e	the smallest time stamp at which $opt_e(t).val$ can be minimized for $t \in [\lambda_e, t_a]$

Table 5.4: Subgraphs of OL.

subgraph ID	♯nodes	♯edges
1	2,001	2,285
2	3,001	3,421
3	4,001	4,577
4	5,001	5,749
5 (original OL)	6,105	7,035

Table 5.5: Subgraphs of TG.

subgraph ID	♯**nodes**	♯**edges**
1	2K	1,335
2	6K	5,306
3	10K	10,235
4	14K	16,113
5 (original TG)	18,263	23,874

Table 5.6: Subgraphs of FLA.

subgraph ID	♯**nodes**	♯**edges**
1	200,000	252,772
2	400,000	506,768
3	600,000	764,006
4	800,000	1,017,156
5 (original FLA)	1,070,376	1,456,400

Chapter 6

Conclusions and Future Work

We conclude this thesis and propose several promising research topics for future work in this chapter.

6.1 Conclusions

This research work investigates three LBSs related spatial queries among moving objects, aiming to propose novel solutions to proximity detection, points of interests recommendation, and cost-optimal time-dependent route queries, respectively. The objectives of these three spatial queries are to (i) present new efficient algorithms that induce low communication cost for proximity detection in road networks, (ii) design a unified framework for POI recommendation, and (iii) find a cost-optimal route in a time-dependent road network, respectively.

We answer the query of road-network proximity detection by proposing two types of solutions. With modern GPS-equipped mobile devices being pervasively utilized, LBSs have a strong appeal to a large number of researchers. Among these LBSs, proximity detection proves to be a typically hot topic. Motivated by various applications like real-world LBSs and a huge number of virtual games, efficient approaches are required to continuously detect proximity among mobile clients. We have studied existing research on proximity detection problems and found that some of these existing research focus on reducing the communication cost while some others focus on reducing the computational cost. In addition, existing research mainly concentrate on proximity detection problems

in Euclidean space rather than road network space and to our best knowledge, no existing works before ours propose self-adjustment algorithms in road networks. That is the motivation for us to develop algorithms with self-adjustment policy that can be used in road network. Our main goals are to design algorithms that detect proximity in road networks with the client-server communication cost minimized by adopting the client-server architecture. We first develop fixed-radius algorithms for the client and server separately, which aim to prune unnecessary update messages sent by the clients to the server, as well as unnecessary probing messages sent by the server to the friend pairs that are guaranteed to be or not to be within proximity. No update messages will be sent unless the client moves beyond its mobile region and no probing messages will be sent unless none of the three pruning lemmas is satisfied. Experiments demonstrate that our fixed-radius solution has a relatively low probing cost but a relatively high update cost when the radius is small, and a relatively low update cost but a relatively high probing cost when the radius is large. Therefore, we develop our second type of solution by utilizing a self-adjusting policy that automatically tunes mobile regions of each client for minimizing the total communication cost. Based on this self-adjustment policy, we present a new self-adjustment algorithm called RRMD (Radius-based Reactive Mobile Detection), which automatically tunes the radius to approach the optimal radius. Experiments show that this policy performs well and reduces the total communication cost greatly compared to the fixed-radius method and other competitors ($\mathrm{RMD_{RN}}$/$\mathrm{CMD_{RN}}$ method). In addition, experiments also show that our self-tuning solution is robust and has high scalability to different parameters such as number of moving objects, average number of friends for each user and proximity threshold, etc.

For the second query related to POI recommendation, we develop a Popularity-Temporal-Geographical framework, namely, PTG-Recommend, for recommending POIs to a user. The framework firstly proposes an algorithm, namely, SEM-DTBJ-Cluster, which clusters and reversely geocodes GPS points for extracting semantic POIs. Afterwards the framework considers the influence of popularity, temporal influence, and the geographical influence of the POIs, and combines them, for the purpose of deriving a unified rec-

ommendation score for each POI for a user. Our PTG-Recommend is the first framework that studies the influences of popularity, temporal and geographical features from GPS trajectories for location recommendation. We carry out experiments to evaluate the effects caused by popularity, temporal features, and geographical features of POIs, respectively. Experiments indicate that the PTG-Recommend framework outperforms the baseline methods with regard to the precision and recall by 20% to 30%.

For the third query, i.e., the cost-optimal time-dependent routing problem, we aim at finding a minimum-cost route that begins from the source node n_s and ends at the destination node n_e, and meanwhile satisfies time and speed constraints in a time-dependent road network. The aforementioned query is named as COTER. We design an approximate ALG-COTER algorithm to solve COTER. This ALG-COTER algorithm first computes the earliest and latest arrival time for each candidate node, which refers to a node that is reachable from n_s and meanwhile can reach n_e; then computes the topological order of these candidate nodes by utilizing the topological sorting algorithm; and subsequently computes the minimum cost functions for each node iteratively by dynamic programming according to their topological order and the recurrence relation formula; and finally backtracks the optimal route R and computes the waiting time for each node in the optimal route. We also analyze the time complexity of the ALG-COTER. We carry out experiments to evaluate the performance of our ALG-COTER algorithm by studying the influences of different parameters on run time. Results of experiments indicate that our algorithms can answer the COTER query efficiently, and are scalable to various parameters which can influence the run time differently.

6.2 Future Work

This research also brings some promising topics for further studies:

6.2.1 Optimal Route Search from A Place to A Recommended POI

6.2.1.1 Motivation

As discussed in Chapter 1 and Chapter 2, POI recommendation and optimal route search have many applications in both the real world and the virtual computer games. Consider the following situation: A group of users want to visit several far-away points of interest. However, as there are many POIs, they plan to visit only one POI with the highest recommendation score. Meanwhile, they need an optimal route so that the total cost on this route from their current place to the POI can be minimized.

The above problem is a combination of POI recommendation and optimal route search. This problem is very common in tourism industry. Therefore, solving such a combined problem is in high demand.

6.2.1.2 Problem statement

We give the definition of this problem as follows: Given a road network G, a group of users U, a set of POIs along the road network, users' POI check-in trajectories, and a source node n_s, our goals are to find: (i) a POI l_i which has the highest recommendation score for user $u_i \in U$; (ii) a cost-optimal route from n_s to POI l_i.

6.2.1.3 Suggested solutions

To answer the above query, we propose the following approaches.

- **Compute user's similarity by checking the proximity relation of two POIs.** If two users always check in the POIs which are within proximity, or even the same POIs, then the similarity between these two users is relatively high. Thus, in this step, we can use the proximity detection method to obtain users' similarity, which is also the basis of computing the recommendation score of each POI.

- **Find the POI l_i which has the highest recommendation score for each user by using our PTG-Recommend framework.** For each user, we use our PTG-Recommend framework to compute the recommendation score for each POI, and

then selects the one whose recommendation score is the highest one with respect to each user.

- **Find the optimal route from n_i to POI l_i for each user u_i.** We can use our ALG-COTER algorithm to compute a cost-optimal route from n_i to POI l_i for each user u_i.

6.2.2 Proximity Detection in Dynamic Road Networks

6.2.2.1 Motivation

Either in reality or the virtual games, the structure of a road network, or the throughput of the road network may vary at different time intervals. We can distinguish two typical categories of events in dynamic networks.

1. Structure changes of the network due to road maintenance or repairing, make us deal with a new road network. For instance, introducing a new route or removing an existing route, introducing a new junction or removing an existing junction, etc.

2. Flow changes of the road network due to rush hours or unexpected events make the road network a dynamic one. For instance, during rush hours, maximum flow may happen and the speed of flows may decrease accordingly. When a flow changes to a halt state, its speed may be reduced to zero; when a flow changes to a starting state, its speed increases again. Another example is the drying up of a flow indicates its density decreases to zero.

In [15], considering that the possible beneficial and counter-productive effects may be caused by enhanced motorist information and that information on network conditions influences the set of routes considered by a driver and also affects the perceived values of the level of service attributes, the authors present the structure of a dynamic model in which newly acquired information may affect pretrip and en-route travel decisions.

In [39], to model the dynamic road network, a *temporal attribute* is defined to describe the state history of the junction or route. It is represented in a form $tp = ((I_i, s_i))_{i=1}^{n}$,

indicating that in time interval I_i, the state of this junction or route is s_i. Here, s_i can be open during normal periods or blocked because of a traffic jam.

In [45], the authors describe processes and events in dynamic road networks which are claimed to contribute to the general research effort toward a generic ontology [1] of dynamic geographic-scale phenomena and its application to the provision of modeling, analysis, and retrieval of data in a spatial-temporal GIS.

In [136], a directed graph $G_r = (V_r, E_r)$, where V_r and E_r denote the sets of nodes and edges, is used to define a *dynamic road network*. The time needed for traversing an edge is dynamic at least in the following two aspects: (i) Time-dependent. Typically, the traffic flow on a road surface varies over days of the week and time of a day, e.g., a road may become crowded in rush hours while be quite smooth at other times; (ii) Location-variant. Different roads have different time-variant traffic patterns. For instance, some streets could still be very fast even in the morning rush. However, the rush hours of a few roads may last for an entire day.

The works above are all focused on the properties of dynamic road network itself, rather than proximity detection problems. Only few works resolve the proximity detection problem in time-aware road networks. The work [69] studies proximity queries in time-dependent road networks using graph embedding technique. In addition, the work [69] aims at reducing the computational cost instead of reducing communication cost. Therefore, there is still a large potential to extend our current work to dynamic road networks.

6.2.2.2 Problem statement

The problem of proximity detection in dynamic road networks can be defined as follows. Suppose $G_T = (V, E)$ represents a dynamic road network, where V and E represent the sets of junctions and routes. At different time intervals, V and E may be different in that some routes or junctions may be blocked during some time intervals. In addition to G_T, a collection of moving users U in G_T, as well as their friendships and the time threshold t_ϵ between them are given, our task is to develop efficient algorithms together with a feasible dynamic road network model to find whether the time distance between

each pair of friends is within t_ϵ. Similarly to our previous work [82], the main objective of addressing this problem is to cut the total communication cost.

6.2.2.3 Suggested solutions

To solve this problem, we propose suggested solutions as follows.

- **Extend current static network model so that it can describe the properties of dynamic networks.** Considering the events that may occur in a dynamic road network, we introduce new attributes, e.g., states of a junction or route, and time interval, into current network model. Thus we can represent different status of a route or junction at different time intervals. Meanwhile, the speeds of a moving object may vary at the same route at different time intervals. When a route is blocked by traffic jams or obstacles, many objects may have to stop and wait, therefore at such time stamps their speeds become zero. We need to take this into consideration when modeling the dynamic road network.

- **Use the metric of time distance to measure proximity instead of network distance.** In dynamic road networks which is time-dependent, the time distance between them, is defined as the time needed for a user to reach his friend from his current position. In dynamic road networks, the speed of each object is dependent upon time, and therefore users care more about the time distance between them rather than network distance between them. Hence, we adopt the metric of time distance to measure the proximity relation instead of network distance. If the time distance between a pair of friends does not exceed the proximity threshold, we say this friend pair is within proximity.

- **Precompute the shortest road network distance and keep a record of the shortest paths between any two junctions in the initial-state road network.** We need to compute the time distance by using the network distance. The network distance changes when the structure of the road network changes. Moreover, the network distance in a dynamic road network can be recomputed on the basis of the network

distance computed on the initial-state road network which refers to the road network at the first time stamp. As a result, we precompute the shortest paths between any two junctions based on the initial-state road network and store the paths for later use.

- **At a new time stamp, when an edge E_i or a node V_i is blocked, recompute the road network distances between the junctions that are affected by E_i or V_i.** We have known all the shortest paths between two junctions in our previous step, and therefore if the state of a route E_i or a junction V_i has changed, all the shortest paths that pass E_i or V_i must be recomputed. Other paths that are not affected by E_i or V_i do not need to be recomputed.

- **Apply update methods and pruning lemmas proposed in current work to this problem to reduce the exchanging messages.** Aiming to reduce the probing cost, we propose two pruning lemmas by using the lower bound and upper bound of the time distance: (i) for objects O_m and O_n, in case the lower bound of their time distance, i.e., $T_{min}(R_m(t), R_n(t))$, is larger than the time threshold T_ϵ, then the time distance between this friend pair must be larger than T_ϵ, thus this friend pair should be pruned; and (ii) for moving objects O_m and O_n, in case the upper bound of their time distance, i.e., $T_{max}(R_m(t), R_n(t))$, is no larger than T_ϵ, then this friend pair must be selected into the result set. Aiming to reduce the update cost, similarly to our previous work [82], we also use region-based update strategy at the client side. As long as the object does not move beyond its mobile region, no updates are needed.

6.2.3 Mining Semantic Patterns From GPS Trajectories

This subsection proposes the problem of mining semantic patterns from GPS trajectories. First we present the motivation as well as the problem statement, and subsequently we give suggested solutions for addressing this problem.

6.2.3.1 Motivation

The increasing pervasiveness and wide utilization of GPS-enabled devices guarantee the availability of a huge amount of GPS data. Mobile devices equipped with GPS are able to generate a great many GPS records that collect continuously changing geolocations with time stamps, and some other information, e.g., direction and speed. As a continuous function from time to space, a client's trajectory can be approximated by the sequence of GPS records for the mobile client. Trajectories contain important semantic information since the interaction between the moving clients and geographical space can be captured by the GPS trajectories.

The fact that GPS data is receiving great attention from scientists and researchers, encourages us to study more about GPS data. A majority of Web sites based on communities enable users to share travel trajectories among each other. There exist several works studying the GPS trajectories. For example, the authors of [133] mine user similarity from semantic trajectories. A semantic trajectory data mining query language, namely, ST-DMQL, is proposed by the study [16]. Another work [4] proposes a framework for semantic trajectory knowledge discovery. Cao $et\ al.$ mine important semantic locations from GPS data in their work [21]. We also propose a framework for POI recommendation from GPS trajectories in our work [81]. However, existing literature does not mine sufficient semantic patterns from GPS trajectories. Patterns are important to users as they provide the "text-understanding" information. Therefore, we aim to consider semantic locations throughout, and discover semantic patterns of moving clients from their GPS trajectories.

6.2.3.2 Problem statement

We give the definition of the problem of mining semantic patterns from GPS trajectories as follows. Given a group of moving users *U*, together with their GPS trajectories *Traj*, our objective is to find the semantic patterns, that is, the patterns of textual semantic information of locations from GPS trajectories. These patterns include the influence of different occupations, different religions, and different social status, etc.

6.2.3.3 Suggested solutions

To address this problem, we propose suggested solutions as follows.

- Identify the GPS points from GPS trajectories.
- Cluster GPS locations from GPS points.
- Translate the GPS trajectories into semantic texts on the basis of trajectory ontology [121, 127]. This step includes finding out the name of the building at the specific location, and the usage of this semantic location, and so on.
- Use the patterns that have been already mined to further help to translate the remaining GPS trajectories into semantic information.

6.2.4 Multi-preference Routing in Dynamic Road Networks

This subsection proposes another promising query problem, i.e., multi-preference routing in dynamic road networks.

6.2.4.1 Motivation

Routing problems have attracted researchers' much attention in the past decades. Existing works like [30, 33, 32] propose customizable route planning algorithms in road networks. There are also some other works [42, 20] presenting algorithms for finding optimal routes. Due to weather conditions, traffic jam, or other unexpected emergency, road networks are more probably dynamic or time-dependent rather than static. Therefore, some works such as [34] and [31] study routing in time-dependent road networks. Currently, as mentioned in Chapter 5, we also propose an ALG-COTER algorithm to find the cost-optimal route in a time-dependent road network.

Because of limited energy resources, people prefer more economic routes rather than an energy costing one. Therefore, saving energy cost is a common objective when dealing with routing problems. Further, nowadays, people enjoy traveling around the world. In such a situation, people tend to design a route with cheaper money cost and more places

of interest. Moreover, if a traveler has limited time for traveling, then she might prefer viewing as many as possible places of interest, with as cheap as possible money cost (usually there is a money budget), without consuming much physical capacity, within limited time.

Routing problem in such circumstances results in a multi-preference routing problem. However, there exists few works studying such kind of multi-preference routing problems. Therefore, researchers are expected to develop algorithms to design a multi-preference route according to users' multiple preference.

6.2.4.2 Problem statement

Given a user u with a money budget b, a limited physical capacity c, a limited time budget T, and a set of places of interest P, as well as a time-dependent road network G_T. The edge length in G_T is independent of time. But the money cost, physical cost, and maximum speed allowed on each edge are all dependent upon time. The objective is to find such a route, along which the user u can view as many as possible places of interest in P, but the money cost does not exceed her money budget b, the physical cost does not exceed her physical capacity c, and the total time does not exceed T.

6.2.4.3 Suggested solutions

To resolve the aforementioned multi-preference routing problem, some suggested approaches are given as follows.

- Model the problem using mathematical linear programming with constraints.
- Develop a dynamic programming algorithm to address this problem. We first define a most-POIs function for each node which denotes the largest number of POIs that a user could view when arriving at this node at time point t, and then derive the recursive relation of the most-POIs functions of this node and its incoming neighbors.
- Design a greedy algorithm with acceptable approximation accuracy to answer this multi-preference route query.

- Evaluate solutions using both the simulated dataset and the real-world dataset.

References

[1] what is protégé? http://protege.stanford.edu/overview/index.html.

[2] P. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *Journal of Computer and System Sciences*, 66(1):207–243, 2003.

[3] H. Aljazzar and S. Leue. K*: A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*, 175(18):2129 – 2154, 2011.

[4] L. O. Alvares, V. Bogorny, B. Kuijpers, J. de Macelo, B. Moelans, and A. T. Palma. Towards semantic trajectory knowledge discovery. *Data Mining and Knowledge Discovery*, 2007.

[5] A. Amir, A. Efrat, J. Myllymaki, L. Palaniappan, and K. Wampler. Buddy tracking–efficient proximity detection among mobile friends. *Pervasive and Mobile Computing*, 3(5):489–511, 2007.

[6] A. Artmeier, J. Haselmayr, M. Leucker, and M. Sachenbacher. The shortest path problem revisited: Optimal routing for electric vehicles. In *KI 2010: Advances in Artificial Intelligence*, pages 309–316. Springer, 2010.

[7] D. Ashbrook and T. Starner. Using gps to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing*, 7(5):275–286, 2003.

[8] B. Badrinath, A. Bakre, T. Imielinski, and R. Marantz. Handling mobile clients: A case for indirect interaction. In *Proceedings of Fourth Workshop on Workstation Operating Systems*, pages 91–97. IEEE, 1993.

[9] D. Barbará and T. Imieliński. Sleepers and workaholics: caching strategies in mobile environments (extended version). *The VLDB Journal*, 4(4):567–602, 1995.

[10] S. Bashash and H. K. Fathy. Cost-optimal charging of plug-in hybrid electric vehicles under time-varying electricity price signals. *IEEE Trans. Intell. Transp. Syst.*, 15(5):1958–1968, 2014.

[11] G. V. Batz, D. Delling, P. Sanders, and C. Vetter. Time-dependent contraction hierarchies. In *ALENEX*, volume 9. SIAM, 2009.

[12] M. Baum, J. Dibbelt, L. Hübschle-Schneider, T. Pajor, and D. Wagner. Speed-consumption tradeoff for electric vehicle route planning. In *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, page 138, 2014.

[13] M. Baum, J. Dibbelt, T. Pajor, and D. Wagner. Energy-optimal routes for electric vehicles. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 54–63. ACM, 2013.

[14] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. *The R*-tree: an efficient and robust access method for points and rectangles*, volume 19. ACM, 1990.

[15] M. Ben-Akiva, A. De Palma, and K. Isam. Dynamic network models and driver information systems. *Transportation Research Part A: General*, 25(5):251–266, 1991.

[16] V. Bogorny, B. Kuijpers, and L. O. Alvares. St-dmql: A semantic trajectory data mining query language. *IJGIS*, 23(10):1245–1276, 2009.

[17] T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.

[18] X. Cai, T. Kloks, and C. Wong. Time-varying shortest path problems with constraints. *Networks*, 29(3):141–150, 1997.

[19] Y. Cai, K. Hua, and G. Cao. Processing range-monitoring queries on heterogeneous mobile objects. In *Proceedings of IEEE International Conference on Mobile Data Management*, pages 27–38. IEEE, 2004.

[20] X. Cao, L. Chen, G. Cong, and X. Xiao. Keyword-aware optimal route search. *Proceedings of the VLDB Endowment*, 5(11):1136–1147, 2012.

[21] X. Cao, G. Cong, and C. S. Jensen. Mining significant semantic locations from gps data. *Proc. of the VLDB Endowment*, 3(1-2):1009–1020, 2010.

[22] E. Chan and H. Lim. Optimization and evaluation of shortest path queries. *The VLDB Journal*, 16(3):343–369, 2007.

[23] E. P. Chan and Y. Yang. Shortest path tree computation in dynamic graphs. *IEEE Transactions on Computers*, 58(4):541–557, 2009.

[24] Y.-J. Chen, K.-T. Chuang, and M.-S. Chen. Coupling or decoupling for knn search on road networks?: a hybrid framework on user query patterns. In *Proc. of CIKM '11*, pages 795–804. ACM, 2011.

[25] C. Cheng, H. Yang, I. King, and M. R. Lyu. Fused matrix factorization with geographical and social influence in location-based social networks. In *AAAI*, volume 12, page 1, 2012.

[26] H. Cho and C. Chung. An efficient and scalable approach to cnn queries in a road network. In *Proceedings of the 31st international conference on Very large data bases*, pages 865–876. VLDB Endowment, 2005.

[27] K. Chu, M. Lee, and M. Sunwoo. Local path planning for off-road autonomous driving with avoidance of static obstacles. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1599–1616, 2012.

[28] A. Civilis, C. S. Jensen, and S. Pakalnis. Techniques for efficient road-network-based tracking of moving objects. *IEEE Trans. Knowl. Data Eng.*, 17(5):698–712, 2005.

[29] A. Das Sarma, S. Gollapudi, M. Najork, and R. Panigrahy. A sketch-based distance oracle for web-scale graphs. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 401–410. ACM, 2010.

[30] D. Delling. *Engineering and augmenting route planning algorithms*. PhD thesis, Karlsruhe Institute of Technology, 2009.

[31] D. Delling. Time-dependent sharc-routing. *Algorithmica*, 60(1):60–94, 2011.

[32] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Customizable route planning. In *Experimental Algorithms*, pages 376–387. Springer, 2011.

[33] D. Delling, M. Holzer, K. Müller, F. Schulz, and D. Wagner. High-performance multi-level routing. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, 74:73–92, 2009.

[34] D. Delling and D. Wagner. Time-dependent route planning. In *Robust and Online Large-Scale Optimization*, pages 207–230. Springer, 2009.

[35] D. Delling and R. F. Werneck. Faster customization of road networks. *SEA*, 13:30–42, 2013.

[36] K. Demestichas, M. Masikos, E. Adamopoulou, S. Dreher, and A. Diaz de Arkaya. Machine-learning methodology for energy efficient routing. In *19th ITS World Congress*, 2012.

[37] B. Ding, J. X. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, pages 205–216. ACM, 2008.

[38] Y. Ding and X. Li. Time weight collaborative filtering. In *Proc. of CIKM*, pages 485–492. ACM, 2005.

[39] Z. Ding and R. Guting. Managing moving objects on dynamic transportation networks. In *Proceedings of 16th International Conference on Scientific and Statistical Database Management*, pages 287–296. IEEE, 2004.

[40] Y. Dong, Y. Yang, J. Tang, Y. Yang, and N. V. Chawla. Inferring user demographics and social strategies in mobile social networks. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 15–24, New York, NY, USA, 2014. ACM.

[41] M. Duckham and L. Kulik. "simplest" paths: Automated route selection for navigation. In *Spatial Information Theory. Foundations of Geographic Information Science*, pages 169–185. Springer, 2003.

[42] J. Eisner, S. Funke, and S. Storandt. Optimal route planning for electric vehicles in large networks. In *AAAI*, 2011.

[43] F. Espinoza, P. Persson, A. Sandin, H. Nyström, E. Cacciatore, and M. Bylund. Geonotes: Social and navigational aspects of location-based information systems. In *Ubicomp 2001: Ubiquitous Computing*, pages 2–17. Springer, 2001.

[44] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.

[45] A. Galton and M. Worboys. Processes and events in dynamic geo-networks. *GeoSpatial Semantics*, pages 45–59, 2005.

[46] J. Gao, J. Yu, R. Jin, J. Zhou, T. Wang, and D. Yang. Outsourcing shortest distance computing with privacy protection. *The VLDB Journal*, 22(4):543–559, 2013.

[47] B. Gedik and L. Liu. Mobieyes: Distributed processing of continuously moving queries on moving objects in a mobile system. *Advances in Database Technology-EDBT 2004*, pages 523–524, 2004.

[48] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012.

[49] R. Geisberger and C. Vetter. Efficient routing in road networks with turn costs. In *Experimental Algorithms*, pages 100–111. Springer, 2011.

[50] G. L. Giller. The statistical properties of random bitstreams and the sampling distribution of cosine similarity. *Available at SSRN 2167044*, 2012.

[51] C. Guo, Y. Ma, B. Yang, C. S. Jensen, and M. Kaul. Ecomark: evaluating models of vehicular environmental impact. In I. F. Cruz, C. A. Knoblock, P. Kröger, E. Tanin, and P. Widmayer, editors, *Proc. of the SIGSPATIAL*, pages 269–278. ACM, 2012.

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

[53] F. Hartmann and S. Funke. Energy-efficient routing: Taking speed into account. In *KI 2014: Advances in Artificial Intelligence*, pages 86–97. Springer, 2014.

[54] K. J. Hee, W. William, S. Benjamin, and B. Gaetano. Extracting places from traces of locations. In *Proc. ACM international workshop on Wireless mobile applications and services on WLAN hotspots*, pages 110–118. ACM, 2004.

[55] M. Holzer, F. Schulz, and D. Wagner. Engineering multilevel overlay graphs for shortest-path queries. *Journal of Experimental Algorithmics (JEA)*, 13:5, 2009.

[56] B. C. Housel and D. B. Lindquist. Webexpress: a system for optimizing web browsing in a wireless environment. In *Proceedings of the 2nd annual international conference on Mobile computing and networking*, MobiCom '96, pages 108–116. ACM, 1996.

[57] H. Hu, J. Xu, and D. L. Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *Proc. of the ACM SIGMOD*, pages 479–490, 2005.

[58] B. Huang, R. L. Cheu, and Y. S. Liew. Gis and genetic algorithms for hazmat route planning with security considerations. *International Journal of Geographical Information Science*, 18(8):769–787, 2004.

[59] S. Ilarri, E. Mena, and A. Illarramendi. Location-dependent query processing: Where we are and where we are heading. *ACM Comput. Surv.*, 42(3), 2010.

[60] S. Ilarri, R. Trillo, and E. Mena. Springs: A scalable platform for highly mobile agents in distributed computing environments. In *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, pages 633–637. IEEE Computer Society, 2006.

[61] G. S. Iwerks, H. Samet, and K. P. Smith. Maintenance of k-nn and spatial join queries on continuously moving points. In *ACM Transactions on Database Systems (TODS)*. ACM, 2006.

[62] J. Jing, A. S. Helal, and A. Elmagarmid. Client-server computing in mobile environments. *ACM Comput. Surv.*, 31(2):117–157, June 1999.

[63] S. Jung and S. Pramanik. An efficient path computation model for hierarchically structured topographical road maps. *Knowledge and Data Engineering, IEEE Transactions on*, 14(5):1029–1046, 2002.

[64] M. Kjærgaard, G. Treu, P. Ruppel, and A. Küpper. Efficient indoor proximity and separation detection for location fingerprinting. In *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, page 1. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[65] G. Kollios, D. Gunopulos, and V. Tsotras. On indexing mobile objects. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 261–272. ACM, 1999.

[66] N. Koudas, B. Ooi, K. Tan, and R. Zhang. Approximate nn queries on streams with guaranteed error/performance bounds. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 804–815. VLDB Endowment, 2004.

[67] H. Kriegel, P. Kröger, P. Kunath, M. Renz, and T. Schmidt. Proximity queries in large traffic networks. In *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, page 21. ACM, 2007.

[68] H. Kriegel, P. Kröger, and M. Renz. Continuous proximity monitoring in road networks. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 12. ACM, 2008.

[69] H.-P. Kriegel, P. Kröger, M. Renz, and F. D. Winter. Proximity queries in time-dependent traffic networks using graph embeddings. In *Proceedings of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 45–54. ACM, 2011.

[70] H.-P. Kriegel, M. Renz, and M. Schubert. Route skyline queries: A multi-preference path planning approach. In *2010 IEEE 26th International Conference on Data Engineering (ICDE)*, pages 261–272. IEEE, 2010.

[71] A. Küpper and G. Treu. From location to position management: User tracking for location-based services. In *Kommunikation in Verteilten Systemen (KiVS) Kurzbeiträge und Workshop*, pages 81–88, 2005.

[72] A. Küpper and G. Treu. Efficient proximity and separation detection among mobile targets for supporting location-based community services. *ACM SIGMOBILE Mobile Computing and Communications Review*, 10(3):1–12, 2006.

[73] T. Kurashima, T. Iwata, T. Hoshide, N. Takaya, and K. Fujimura. Geo topic model: joint modeling of user's activity area and interests for location recommendation. In *Proc. of WSDM*, pages 375–384. ACM, 2013.

[74] J. Leape. The london congestion charge. *The Journal of Economic Perspectives*, pages 157–176, 2006.

[75] A. Leonhardi and K. Rothermel. Protocols for updating highly accurate location information. *Geographic Location in the Internet*, pages 111–141, 2002.

[76] Q. Li, H. Fan, X. Luan, B. Yang, and L. Liu. Polygon-based approach for extracting multilane roads from openstreetmap urban road networks. *International Journal of Geographical Information Science*, 28(11):2200–2219, 2014.

[77] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W.-Y. Ma. Mining user similarity based on location history. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 34. ACM, 2008.

[78] X. Li, W. Szeto, and M. OMahony. Modeling time-dependent tolls under transport, land use, and environment considerations. In *Applications of Advanced Technology in Transportation (2006)*, pages 852–857. ASCE, 2006.

[79] M. Lichman and P. Smyth. Modeling human location data with mixtures of kernel densities. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 35–44, New York, NY, USA, 2014. ACM.

[80] B. Liu, Y. Fu, Z. Yao, and H. Xiong. Learning geographical preferences for point-of-interest recommendation. In *Proc. of SIGKDD*, pages 1043–1051. ACM, 2013.

[81] Y. Liu and H. S. Seah. Points of interest recommendation from gps trajectories. *International Journal of Geographical Information Science*, 2015.

[82] Y. Liu, H. S. Seah, and G. Cong. Efficient proximity detection among mobile objects in road networks with self-adjustment methods. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL'13, pages 124–133. ACM, 2013.

[83] D. Luxen and C. Vetter. Real-time routing with openstreetmap data. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 513–516. ACM, 2011.

[84] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[85] W. Matthew Carlyle and R. Kevin Wood. Near-shortest and k-shortest simple paths. *Networks*, 46(2):98–109, 2005.

[86] K. Mehlhorn and M. Ziegelmann. Resource constrained shortest paths. *Lecture Notes in Computer Science*, 1879:326–337, 1999.

[87] B. L. Milenova and M. M. Campos. O-cluster: Scalable clustering of large high dimensional data sets. In *ICDM 2003*, pages 290–297. IEEE, 2002.

[88] M. F. Mokbel, X. Xiong, and W. G. Aref. Sina: scalable incremental processing of continuous queries in spatio-temporal databases. In *Proc. of the ACM SIGMOD*, pages 623–634, 2004.

[89] A. Monreale, F. Pinelli, R. Trasarti, and F. Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *Proc. of SIGKDD*, pages 637–646. ACM, 2009.

[90] K. Mouratidis, D. Papadias, S. Bakiras, and Y. Tao. A threshold-based algorithm for continuous monitoring of k nearest neighbors. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1451–1464, 2005.

[91] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou. Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 634–645. ACM, 2005.

[92] K. Mouratidis, M. Yiu, D. Papadias, and N. Mamoulis. Continuous nearest neighbor monitoring in road networks. Very Large Data Bases Conference (VLDB), 2006.

[93] J. Myllymaki and J. Kaufman. High-performance spatial indexing for location-based services. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 112–117. ACM, 2003.

[94] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik. The v*-diagram: a query-dependent approach to moving knn queries. *Proc. VLDB Endow.*, 1:1095–1106, August 2008.

[95] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 802–813. VLDB Endowment, 2003.

[96] S. Prabhakar, Y. Xia, D. Kalashnikov, W. Aref, and S. Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Transactions on Computers*, 51(10):1124–1140, 2002.

[97] M. Qu, H. Zhu, J. Liu, G. Liu, and H. Xiong. A cost-effective recommender system for taxi drivers. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 45–54, New York, NY, USA, 2014. ACM.

[98] A. Rae, V. Murdock, A. Popescu, and H. Bouchard. Mining the web for points of interest. In *Proc. of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 711–720. ACM, 2012.

[99] P. Reiher, J. Popek, M. Gunter, J. Salomone, and D. Ratner. Peer-to-peer reconciliation based replication for mobile computers. In *European Conference on Object Oriented Programming, Second Workshop on Mobility and Replication*. Citeseer, 1996.

[100] R. Richa, M. Balicki, R. Sznitman, E. Meisner, R. Taylor, and G. Hager. Vision-based proximity detection in retinal surgery. *IEEE Transactions on Biomedical Engineering*, 59(8):2291, 2012.

[101] P. Rigaux, M. Scholl, and A. Voisard. Spatial databases with application to gis. *SIGMOD Record*, 32(4):111, 2003.

[102] G. Samara and A. Pitsillides. Client/intercept: a computational model for wireless environments. In *ICT97, International Conference on Telecommunications*, 2-4 April 1997.

[103] J. Schiller and A. Voisard. *Location-based services*. Morgan Kaufmann, 2004.

[104] F. Schulz, D. Wagner, and C. Zaroliagis. Using multi-level graphs for timetable information in railway systems. In *Algorithm Engineering and Experiments*, pages 43–59. Springer, 2002.

[105] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. 1987.

[106] S. . Shekhar. Experiences with evacuation route planning algorithms. *International Journal of Geographical Information Science*, 26(12):2253–2265, 2012.

[107] J. Sneyers, T. Schrijvers, and B. Demoen. Dijkstra's algorithm with Fibonacci heaps: An executable description in CHR. *In 20th Workshop on Logic Programming (WLP' 06)*, 6:182–191, 2006.

[108] X. Song, Q. Zhang, Y. Sekimoto, and R. Shibasaki. Prediction of human emergency behavior and their mobility following large-scale disaster. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 5–14, New York, NY, USA, 2014. ACM.

[109] Y.-y. Song, E.-j. Yao, T. Zuo, and Z.-f. Lang. Emissions and fuel consumption modeling for evaluating environmental effectiveness of its strategies. *Discrete Dynamics in Nature and Society*, 2013.

[110] C. Spyrou, G. Samaras, E. Pitoura, and P. Evripidou. Wireless computational models: Mobile agents to the rescue. In *Proceedings of the 10th International Workshop on Database and Expert Systems Applications*, pages 127–133. IEEE, 1999.

[111] C. Spyrou, G. Samaras, E. Pitoura, and P. Evripidou. Mobile agents for wireless computing: the convergence of wireless computational models with mobile-agent technologies. *Mobile Networks and Applications*, 9(5):517–528, 2004.

[112] S. Storandt. Quick and energy-efficient routes: computing constrained shortest paths for electric vehicles. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 20–25. ACM, 2012.

[113] Y. Sun, T. F. L. Porta, and P. Kermani. A flexible privacy-enhanced location-based services system framework and practice. *IEEE Trans. Mob. Comput.*, 8(3):304–321, 2009.

[114] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining*, volume 6. Addison-Wesley Longman Publishing Co., 2006.

[115] T. Tielert, D. Rieger, H. Hartenstein, R. Luz, and S. Hausberger. Can v2x communication help electric vehicles save energy? In *12th International Conference on ITS Telecommunications*, pages 232–237, 2012.

[116] G. Treu and A. Küpper. Efficient proximity detection for location based services. In *Workshop on Positioning, Navigation and Communication (WPNC)*, 2005.

[117] G. Treu, T. Wilder, and A. Küpper. Efficient proximity detection among mobile targets with dead reckoning. In *Proceedings of the 4th ACM international workshop on Mobility management and wireless access*, MobiWac '06, pages 75–83. ACM, 2006.

[118] H. Wang, L. Fu, Y. Zhou, and H. Li. Modelling of the fuel consumption for passenger cars regarding driving characteristics. *Transportation Research Part D: Transport and Environment*, 13(7):479 – 482, 2008.

[119] H. Wang, M. Terrovitis, and N. Mamoulis. Location recommendation in location-based social networks using user check-in data. In *Proc. of SIGSPATIAL/GIS*, pages 364–373. ACM, 2013.

[120] Y. Wang, Y. Zheng, and Y. Xue. Travel time estimation of a path using sparse trajectories. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 25–34, New York, NY, USA, 2014. ACM.

[121] R. Wannous, J. Malki, A. Bouju, and C. Vincent. Modelling mobile object activities based on trajectory ontology rules considering spatial relationship rules. *Studies in Computational Intelligence*, pages 249–258, 2013.

[122] S. Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361, 2002.

[123] O. Wolfson, P. Sistla, S. Dao, K. Narayanan, and R. Raj. View maintenance in mobile computing. *ACM Sigmod Record*, 24(4):22–27, 1995.

[124] L. Xiang, Q. Yuan, S. Zhao, L. Chen, X. Zhang, Q. Yang, and J. Sun. Temporal recommendation on graphs via long-and short-term preference fusion. In *Proc. of SIGKDD*, pages 723–732. ACM, 2010.

[125] X. Xiong, M. F. Mokbel, and W. G. Aref. Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *Proc. of the ICDE*, pages 643–654, 2005.

[126] Z. Xu and A. Jacobsen. Adaptive location constraint processing. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 581–592. ACM, 2007.

[127] Z. Yan and D. Chakraborty. Semantics in mobile sensing. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 4(1):1–143, 2014.

[128] Y. Yang, H. Gao, J. X. Yu, and J. Li. Finding the cost-optimal path with time constraint over time-dependent graphs. *Proceedings of the VLDB Endowment*, 7(9), 2014.

[129] M. Ye, P. Yin, W.-C. Lee, and D.-L. Lee. Exploiting geographical influence for collaborative point-of-interest recommendation. In *Proc. of SIGIR*, pages 325–334. ACM, 2011.

[130] Y. Ye, Y. Zheng, Y. Chen, J. Feng, and X. Xie. Mining individual life pattern based on location history. In *MDM'09*, pages 1–10. IEEE, 2009.

[131] H. Yin, Y. Sun, B. Cui, Z. Hu, and L. Chen. Lcars: A location-content-aware recommender system. In *Proc. of SIGKDD*, pages 221–229. ACM, 2013.

[132] J. J.-C. Ying, W.-C. Lee, T.-C. Weng, and V. S. Tseng. Semantic trajectory mining for location prediction. In *Proc.of SIGSPATIAL/GIS*, pages 34–43. ACM, 2011.

[133] J. J.-C. Ying, E. H.-C. Lu, W.-C. Lee, T.-C. Weng, and V. S. Tseng. Mining user similarity from semantic trajectories. In *Proc. of SIGSPATIAL/GIS*, pages 19–26. ACM, 2010.

[134] M.-L. Yiu, L.-H. U, S. Šaltenis, and K. Tzoumas. Efficient Proximity Detection among Mobile Users via Self-Tuning Policies. In *Proc. of the VLDB*, 2010.

[135] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *Proc. of the 21st ICDE*, pages 631–642, 2005.

[136] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 99–108. ACM, 2010.

[137] Q. Yuan, G. Cong, Z. Ma, A. Sun, and N. M. Thalmann. Time-aware point-of-interest recommendation. In *Proc. of SIGIR*, pages 363–372. ACM, 2013.

[138] O. R. Zaïane, A. Foss, C.-H. Lee, and W. Wang. On data clustering analysis: Scalability, constraints, and validation. In *Advances in Knowledge Discovery and Data Mining*, pages 28–39. Springer, 2002.

[139] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. Lee. Location-based spatial queries. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 443–454. ACM, 2003.

[140] R. Zhang, D. Lin, K. Ramamohanarao, and E. Bertino. Continuous intersection joins over moving objects. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 863–872. IEEE, 2008.

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

[142] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining correlation between locations using human location history. In *GIS*, pages 472–475, 2009.

[143] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In *WWW*, pages 791–800, 2009.

[144] C. Zhou, N. Bhatnagar, S. Shekhar, and L. Terveen. Mining personally important places from gps tracks. In *ICDE 2007*, pages 517–526. IEEE, 2007.

[145] C. Zhou, D. Frankowski, P. Ludford, S. Shekhar, and L. Terveen. Discovering personal gazetteers: an interactive clustering approach. In *Proc. of the 12th annual ACM international workshop on Geographic information systems*, pages 266–273. ACM, 2004.

[146] A. Zimdars, D. M. Chickering, and C. Meek. Using temporal data for making recommendations. In *Proc. of UAI*, pages 580–588. Morgan Kaufmann Publishers Inc., 2001.

Author's Publications

1. Yaqiong Liu, Hock Soon Seah. "Cost-Optimal Time-dEpendent Routing with Time and Speed Constraints in Directed Acyclic Road Networks." International Journal of Information Technology & Decision Making. Revision.

2. Yaqiong Liu, Hock Soon Seah. "Constrained energy-efficient routing in time-aware road networks." Geoinformatica. Revision.

3. Yaqiong Liu, Hock Soon Seah. "Points of Interest Recommendation From GPS Trajectories." Volume 29, Issue 6, International Journal of Geographical Information Science (IJGIS). Accepted, pp. 953-979.

4. Yaqiong Liu, Hock Soon Seah, Gao Cong. "Efficient Proximity Detection among Mobile Objects in Road Networks with Self-adjustment Methods." In 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2013), pp. 124-133.

5. Yaqiong Liu, Hock Soon Seah, Ying He, Juncong Lin, Jiazhi Xia. "Sketch Based Image Deformation and Editing with Guaranteed Feature Correspondence." VR-CAI 2011, pp. 141-148.

6. Yaqiong Liu, Hock Soon Seah, Ying He, Juncong Lin, Jiazhi Xia. "Sketch Based Image Deformation and Editing with Guaranteed Feature Correspondence." International Journal of Virtual Reality.