UNIVERSITY OF CALGARY

Recommending Profitable Taxi Travel Routes based on Big Taxi Trajectory Data

by

Wenxin Yang

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN GEOMATICS ENGINEERING

CALGARY, ALBERTA

DECEMBER, 2015

ABSTRACT

Recommending potential profitable routes to reduce the cruising distance of taxis is an active research topic.

This thesis first introduces a temporal probability grid network generated from taxi trajectories, where each grid has been assigned two temporal properties: propability and capacity. Two profitable route recommendation algorithms are proposed: the Shortest Expected Cruising Route (SECR) and Adaptive Shortest Expected Cruising Route (ASECR) algorithms. The ASECR algorithm is an extension of SECR that updates profitable routes constantly, and renews the temporal probability grid network dynamically. To handle large amounts of trajectory data and improve the efficiency of constantly updating the routes, a new data structure kdS-tree with a MapReduce model is proposed.

Case studies compare the time and distance performances of the ASECR and SECR algorithms with two other methods, i.e. the LCP method and the baseline method. These comparisons demonstrate the effectiveness and efficiency of the two proposed algorithms.

ACKNOWLEDGEMENTS

This thesis would not have been possible without support from many people. First I would like to express my sincere gratitude to my graduate supervisor, Dr. Xin Wang, for her continuous providing direction, guidance and reviewing my research work. I am most grateful to Natural Sciences and Engineering Research Council of Canada IPS scholarship for supporting me doing the research and Shenzhen Institutes of Advanced Technology for providing us the GPS data. Last but not least, I also like to acknowledge my family and friends for their encouragement and moral support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES AND ILLUSTRATIONS

LIST OF SYMBOLS, ABBREVIATIONS AND NOMENCLATURE

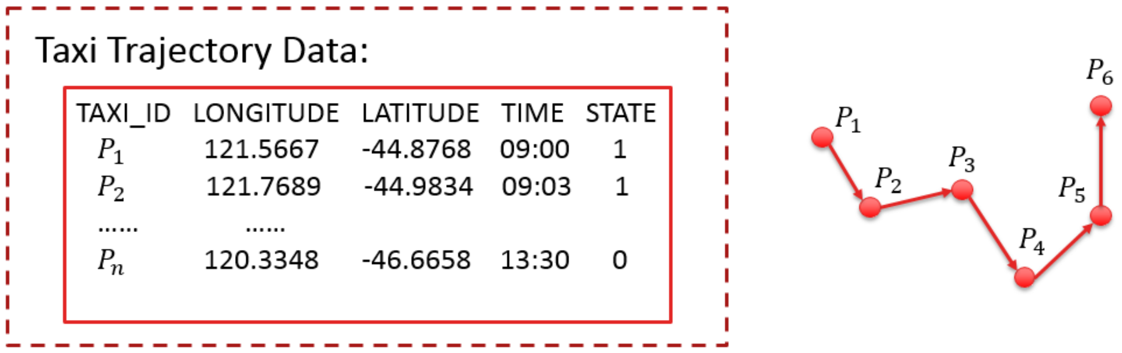
GPS	Global Positioning System
GIS	Geographic Information System
SECR	Two-steps Floating Catchment Area
ASECR	Adaptive Shortest Expected Cruising Route
t	Taxi
γ^t	GPS-Reading
g	Grid
GN	Grid Network
$P(T)$	Distance Performance

CHAPTER ONE: INTRODUCTION

1.1 Background

Taxis play an important role in modern public transportation networks, as they provide fast, secure and personalized travel for passengers. However, a 2006 study reported that taxis spend 35% to 60% of their time cruising along the roads looking for passengers [2]. The reduction of taxis' cruising distances and, thus, energy consumption pose an urgent challenge.

Recently, taxis have been equipped with GPS in many large cities, such as New York and Beijing. These systems can report the taxi's present location, timestamp and status (occupied or cruising) on a certain frequency. Therefore, a large amount of GPS trajectory data with occupancy information (as shown in Figure 1-1) can be generated every day. With the advent of these GPS devices located on taxis, better location-based services based on taxi GPS trajectory data can be provided to improve taxis' performance [3] [4] [5] [6].



Taxi Trajectory Data:

TAXI_ID	LONGITUDE	LATITUDE	TIME	STATE
P_1	121.5667	-44.8768	09:00	1
P_2	121.7689	-44.9834	09:03	1
……	……			
P_n	120.3348	-46.6658	13:30	0

Figure 1-1. Illustration of a GPS reading and trajectory

However, a few challenges exist when using GPS trajectories to improve taxi performance. Since GPS readings are typically noisy, because of electronics, view blockage, or multiple object hitting before reaching the GPS receiver, the locations from the GPS readings usually do not match well with the existing road network. Figure 1-2 shows an example of typical GPS readings on a road network. The green points represent the noisy GPS readings and the blue lines represent the associated polylines on the road network. It is quite difficult to tell which road(s) the taxi is moving on based on these few GPS readings without looking at the entire trajectory.

Moreover, the huge volume of nodes (intersections) and edges (road segments) on the existing road networks make any operation on it very time-consuming. For example, the road networks used in North America already have around 20,000,000 nodes [7], which results in a great deal of processing time. It is impractical to keep taxis waiting such a long time for an algorithm to generate a profitable travel route. To the best of our knowledge, most current research studies only calculate routes heuristically in order to save computing time in large road networks. And as a result, the shortest routes possible are not always presented.

Also, existing recommending methods focus on the suggestion of just one hotspot to the cruising taxi. For example, the five blue dots in Figure 1-3 represent possible hotspots after a taxi drops off a passenger; and, the solution is to output one of them as the best choice at the moment. This method is not the global optimum, as a taxi may pick up a passenger after driving through many hotspots. Although each hotspot is the best recommendation of each step, the recommended routes of the hotspot search may not be the best selection. A good recommendation should be the global optimum, comparing with all the possible route recommendations for the taxi, not a local optimal on each step.

Furthermore, these recommending methods fail to address the route recommendation balance problem with a group of competing taxis. Without a taxi-passenger balance assignment mechanism, the recommended routes will result in poor driving efficiency, due to the unnecessary competition of taxis.

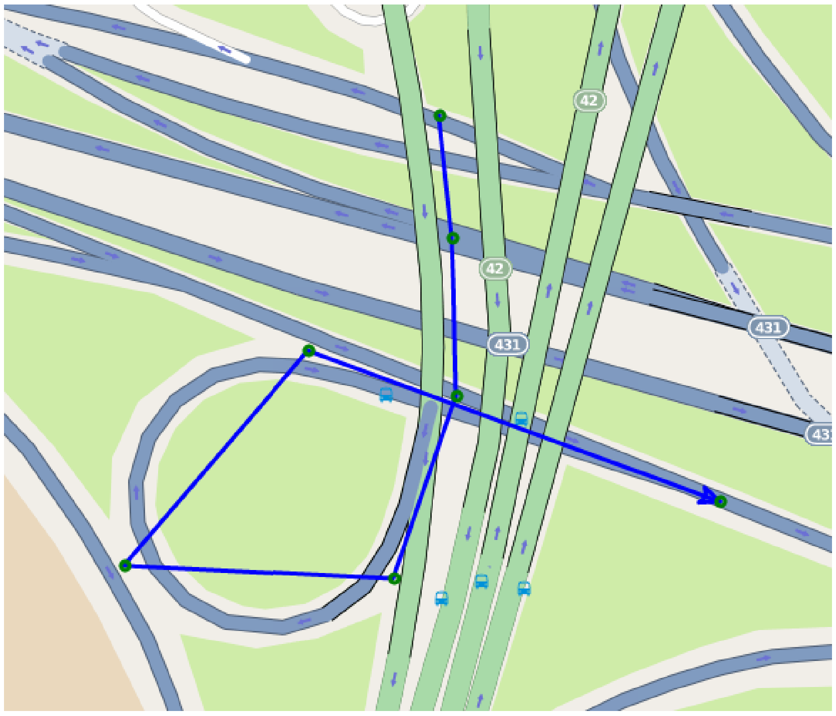


Figure 1-2. Trajectory of GPS Readings on Road Network [8]

Taxi trajectories are recorded in large temporal and spatial datasets, which often exceed the main memory of most stand-alone computers. This prevents the efficient use of traditional centralized technologies. For example, statistics revealed that there are over

70,000 taxis operating in Beijing. These taxis cover every corner of the city 24 hours a day and produce huge volumes of trajectories; for example, the size of one taxi's daily trajectories with a 5-second interval can be around 4 GB [9]. However, most current methods are based on centralized technology and run on just one machine. With these huge volumes of trajectory data, the processing, summary and utilization of passenger information to calculate the recommended routes of taxis is very challenging.

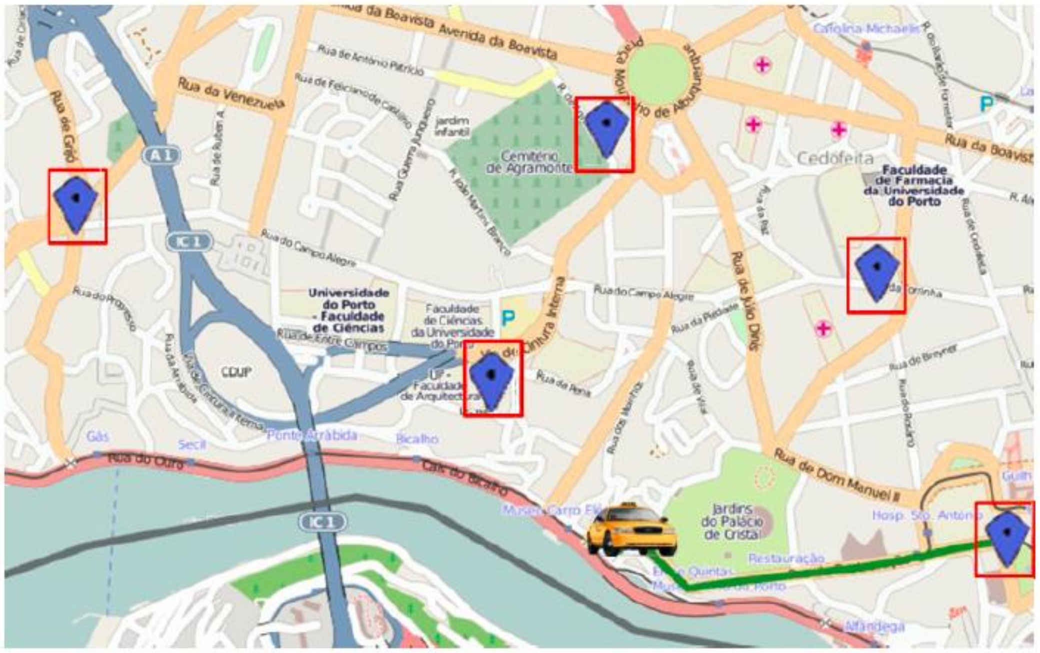


Figure 1-3. Hotspots Choice Problem [10]

1.2 Objectives

The research goal of this thesis is the recommendation of routes for a set of taxis with the shortest cruising distances based on taxi trajectories. Three research topics are investigated: 1) the generation of a road network that can be used to calculate profitable routes within acceptable computation complexity, 2) the provision of a personalized route for each taxi, and 3) the handling of the large amount of trajectory data.

Traditional road networks are represented as graphs, i.e. a collection of nodes (junctions) and edges (road segments), where each edge connects two nodes. The recommended routes for taxis are computed in road networks. Since the road network contains millions of road junctions, which making a reasonable timely response impractical. There is, therefore, considerable interest in the development of more efficient road networks. In this research, we focus on building a new road network, pruning unnecessary nodes that have no probability of potential passengers based on taxis' historical pickup information. This new road network will not only contain the location information of potential passengers, but will also be capable of reducing the calculation time for generating profitable travel routes.

The locations of passengers are unknown, and usually the number of potential passengers varies in the same area over time. Simply sending taxis to a predicted location that may have passengers, without taking into consideration the time period, may result in non-profitable cruising. Thus, a model to predict the spatio-temporal distribution of passenger demand is needed. Moreover, there is usually a group of taxis waiting for recommended routes at the same time; and, if the route recommendation is not customized for the individual taxi, a large number of cruising taxis may be sent to the same location to compete for the same group of passengers. The provision of a personalized route for each taxi without impacting the passenger/taxi balance is, therefore, an important issue. In this thesis, algorithms that provide spatio-temporal travel routes are developed, and taxis can drive the recommended route until a passenger is picked up. The algorithms are also aimed at achieving an approximate global optimum, in order to achieve more profit for a group of taxis by considering the passenger/taxi balance.

The large volumes of trajectory data usually exceed the computation capacity of traditional centralized methods. This requirement has encouraged the research of parallel computing models that can provide a potential for scaling data processing. The MapReduce framework is considered in this research, due to its known quality and simplicity. However, given that the default split strategy in MapReduce may corrupt the data integrity of ordered data, this framework does not directly support trajectory data, which consists of ordered data elements whose time dimension is monotonously increasing and whose latter location is dependent on the former one. Therefore, in order to solve this problem, this research proposes a data structure – the kdS-tree – to classify data in advance. The main goal is to prove that the proposed algorithm can also be deployed on a distributed platform and that a paralleled route recommendation method is not only possible, but simple and productive.

1.3 Contributions

The main contributions of this thesis are:

1. A temporal probability grid network is built by modeling the potential profitability of locations based on taxis' historical GPS data. The grid network separates the region of interest into small grids. Each grid of the grid network includes two temporal properties: the probability of finding a passenger, and the capability of accommodating the cruising taxis.

2. A profitable grid assignment algorithm is proposed based on the spatial and temporal factors of taxis. The assignment algorithm distributes the potential profitable grids among taxis based on the probabilities and capacities of grids. The assignment also balances the workload among a group of taxis.
3. The concept of the shortest expected cruising distance is proposed. The shortest expected cruising distance indicates the potential cruising distance from the taxi to potential passenger based on the probability of picking up a passenger at a location. This concept resulted in the development of the Shortest Expected Cruising Route (SECR) algorithm.
4. The SECR algorithm is extended to the Adaptive Shortest Expected Cruising Route (ASECR) algorithm. ASECR can dynamically recommend the route by finding the most potentially profitable locations based on the current time and the taxi's movement. In addition, the temporal probability grid network will also be updated with different time stamps. A new data structure – the kdS-tree – is proposed to support the efficient updating of the k nearest profitable grids.
5. A MapReduce method along with the new kdS-tree data structure is implemented in the ASECR algorithm to efficiently handle the large amounts of trajectory data and to support the efficient updating of the k nearest profitable grids.

1.4 Organization of the Thesis

This thesis is organized as follows: Chapter 2 provides a literature review on the existing route recommendation systems. Methods of trajectory mining and route recommendation are also described. Chapter 3 introduces the problem setup, defines the notations and describes the implementation of two profitable route recommendation algorithms: the Shortest Expected Cruising Route (SECR) and Adaptive Shortest Expected Cruising Route (ASECR) algorithms. The kdS-tree data structure for improving the efficiency of querying potential pickup locations is also proposed. Chapter 4 explains how to use a MapReduce method to handle the large amounts of taxi trajectory data. Chapter 5 presents case studies. The thesis conclusion and future research directions are discussed in Chapter 6.

CHAPTER TWO: LITERATURE REVIEW

This chapter reviews some of the existing work that closely related to the research in this thesis. Section 2.1 introduces GPS trajectory mining. Section 2.2 presents a variety of route recommendation technologies based on trajectory datasets. Finally, Section 2.3 introduces several spatial temporal data structures as well as a popular parallel framework -- MapReduce which deals with massive-scale data problem.

2.1 Trajectory Mining

GPS (Global Positioning System) technology has many applications. It can be embedded into many mobile devices to track the movement of mobile objects, such as taxis, buses and people. Figure 2-1 is an example of a GPS trajectory dataset in a road network, where the GPS trajectory is depicted as a red triangle along the route in blue.

Nowadays, GPS-equipped taxis are regarded as an ideal data source for uncovering the pick-up pattern for taxis, so mining knowledge from taxi GPS trajectories has become a hot topic. The studies of mining GPS trajectories mainly focus on three areas: 1) efficiency query methods for moving objects' trajectories [11] [12] [13]; 2) techniques of trajectory analyzing, including calibration [14], classification [15], clustering [16] [17] and anomaly detection [18]; 3) real world applications based on GPS trajectories mining techniques, like [19] detected flawed road segments in the view of urban planning according to taxi trajectories.

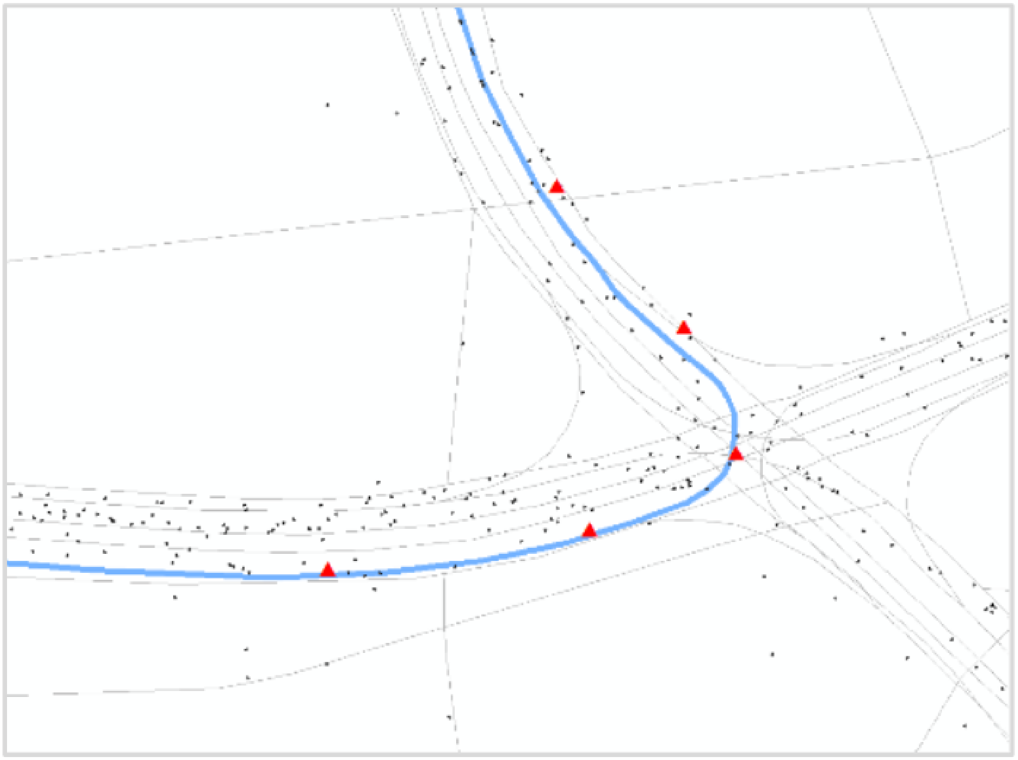


Figure 2-1. An example of GPS Trajectory [20]

2.1.1 Pattern Queries

Wolfson, et al. [11] studied the geometric properties of moving objects' trajectories with proposing a new index structure, the Moving Objects in Networks Tree (MON-Tree) to efficiently store and retrieve moving objects. The index structure could store complete trajectories and be capable to query the moving objects. Besides, two query methods were introduced: range query and window query, which can be answered by MON-Tree efficiently. Both queries took a spatial-temporal window as an argument and differed on their results: the range query returns all objects whose movement overlaps the query window, and on the other side the window query is more precise and returns only the pieces of the objects' trajectory that overlaps the query window. They have experimentally evaluated the proposed index structure against other index structure for capable of indexing moving objects in networks and the results showed their MON-Tree index structure outperformed others. On the other hand, Güting, et al. [12] considered the semantics and background geographic information of trajectories to find trajectory patterns. Trajectories can be classified according to their spatial locations, time stamp, velocity and direction of movement based on existing organized data. By modeling traffic data in a Moving Object Database (MOD), useful data about the movement of vehicles can be further extracted, which is useful in predicting situations, such as “traffic congestion” or “traffic accident” and to provide other options, such as “alternative route” or “recommended routes”. In [13], a general model for semantic trajectories was proposed and a concept of stops and moves was introduced. They explored how conceptual modeling could provide applications with direct support of trajectories. A specific concern is used to enrich trajectories information with semantic annotations which allowing users to add semantic data to specific section of the trajectory.

However, all those models are restricted to a specific application domain, for example: trajectory relationships are only related to vehicles and roads, they did not consider the requirement from the passengers nor the competition between taxis.

2.1.2 Trajectory Classification

Liu, et al. [21] proposed mobility-based clustering algorithm, the key idea is that sample objects are seen as “sensors” to perceive the vehicle’s congestion degree in nearby areas using their instant movement. As such movement of samples is naturally incorporated. Several key factors beyond the vehicle crowdedness had been identified and techniques to compensate these effects are proposed. They evaluated the performance of mobility-based clustering based on real traffic situations. With 0.3% sample object set the mobility-based clustering can measure the spot crowdedness with an accuracy of 61.3% at a certain site, compared with that of 3.3% under the same scenario by the traditional density-based approach UMicro. Lee, et al [22] presented a new partition-and-group framework for clustering trajectories, which partitioned a trajectory into a set of line segments, and then grouped similar line segments together into a cluster. Based on this partition-and-group framework, they developed a trajectory clustering algorithm, which are called TRACLUS algorithm. The main advantage of TRACLUS is the discovery of common sub-trajectories from a trajectory database. Partitioning and grouping are the two important parts of this trajectory clustering algorithm. For the first part, they presented a formal trajectory partitioning algorithm using the minimum description length (MDL) principle. For the second part, they presented a density-based line-segment clustering algorithm. Experimental results demonstrate that TRACLUS effectively identified common sub-trajectories from real trajectory data as clusters.

However, all above methods are only considered the spatial information, in practical, temporal information is also important since the spatial information always changes along with the different time.

2.1.3 Trajectory Pattern Applications

How to improve the schedule of taxi fleet has been studied in the operational research [4][86][87]. For example, Meyer and Wolfe [4] defined two types of strategies in the dispatching market: dispatching strategy and idle-time strategy. The dispatching strategy analyzed how to choose taxi to server the given call while idle – time strategy determines how to distribute taxis to meet passengers' future demands. They estimated waiting time by utilizing Monte Carlo techniques for the complex case when passengers' demands are unevenly distribute in the map.

Since the advance of human civilization encouraged the improvement of the transportation industry, the knowledge discovered in trajectory mining area are widely used in trajectory pattern applications. Zheng, et al. [23] detected modelled trajectories pattern by mining the GPS trajectories of urban areas' taxis. They first partitioned the city into some disjoint regions. Then, the city-wide traffic of taxis of each day is modelled by using a matrix of regions. Each item in the matrix is consisted as a set of features representing the effectiveness of the connection between two different regions. The values of these features were derived from the taxi trajectory that passing through two regions. Later, the salient region pairs which having heavy traffic over the capacity of the existing connections between them are detected. The region pairs frequently detected across many days will be regarded as the flawed planning. At the same time, they evaluated the method by using a series of large-scale real GPS trajectories, and justified the effectiveness of it.

2.2 Route Recommendation

Route recommendation is an important application that leverages the knowledge discovered from historical trajectories, which includes: 1) hotspot recommendations; 2) the fastest route recommendation; 3) the shortest route recommendations and 4) traffic-analysis-based route recommendations. These work either by focusing on identifying hotspots or by combining popular route fragments.

2.2.1 Hotspot Recommendation

A useful concept in discovering valuable knowledge from a GPS trajectory is applying hotspots. In this thesis research domain, hotspots are referred as areas with higher probabilities of passengers than other ones. The identification and analysis of hotspots are useful for taxi route recommendations.

Chang, et al. [24] mined historical data to predict demand distributions with respect to the contexts of time. In their study, hotspots were extracted from large amounts of pick up points using clustering algorithm and then recommended to taxi driver based on the hotness of the hotspots. Similarly, Li, et al. [25] focused on predicting human movements by uncovering patterns of passenger pick-ups quantity (PUQ) from hotspots. An adaptive watershed-based hotspot extractionalgorithm was proposed to cluster the pick-up/set-down events of taxi passengers. An improved ARIMA was proposed to predict the spatial-temporal trend of passengers in a hotspot. The improved ARIMA combines ARIMA with a prior distribution of pick-up values, and achieves better prediction accuracy than the other methods. Li, et al. [26] discussed whether a taxi driver should remain in a local area (waiting) or travel a longer distance (hunting) to find a new passenger. They studied the passenger finding strategy for taxi drivers in Hangzhou, China. In their work, the triples of (location, time, and strategy) were used as the key features to build a passenger finding strategy. Then they used L1-Norm Support Vector Machines (L1-norm SVM) to determine whether the driver should wait for a passenger or keep cruising based on the selected patterns. The experiments showed the taxi performance could achieve a prediction accuracy of 85.3%. However, all above methods do not recommend routes for pick-ups.

2.2.2 Sequence of Hotspots Recommendation

Current research in this field is focused on how to recommend a sequence of hotspots to maximize the probability of picking up a passenger with a shorter cruising distance.

Ge, et al. [1] developed a mobile recommender system for taxi drivers which recommending a sequence of pick-up points so as to maximize a taxi driver's profit. They estimated the probability of pick-up events for each candidate point, and then proposed two algorithms, which are called LCP and SkyRoute, to find a route with minimal potential travel distance before finding a passenger. Specifically, for the length of suggested route L, the LCP algorithm enumerates all the L-length sub-routes from all pick-up points and recommends the best route among these candidate routes to the taxi driver. In addition, they also observed that recommended routes are composed of sub-routes and different routes can cover the same sub-routes. Based on the observation, SkyRoute algorithm is proposed to efficiently compute the candidate routes by pruning some candidate routes, which are comprised of the dominated sub-routes, at a very early stage. Then the search space can be significantly reduced, since lots of candidate routes which contain the dominated sub-routes could be discarded from further consideration as skyline routes.

Hu, et al. [27] proposed a pick-up tree based route recommendation system to minimize the traveling distance without a passenger. They first applied a clustering method to the GPS trajectory to identify taxi states. At the same time, the algorithm discovered potential hotspots which have higher pick-up probabilities and made the hotspot as the centroid. Then, based on the skyline, the system constructed a pick-up tree which took the current position as its root node and connected all the centroids. A probability model was proposed that estimated the weight of every route and the weighted Round-Robin recommendation method for the set of taxis. The experiment showed that the proposed recommendation model effectively reduces the cruising distance before picking up passengers.

A dynamic taxi dispatch system for smart cities was proposed by Zou, et al. [28]. In the system, a dynamic probabilistic model had been established, which considered the impact of time on passenger appearance and the effect of different vacant taxis' traveling routes on each other's pick-up probability. Specifically, a novel feedback system was introduced into the system, which updated information about taxis picking up passengers in certain hotspots, then adjusting the system probabilistic model based on the updated information. Moreover, extensive trace-driven simulations based on large-scale real-world GPS data demonstrated the adequate performance of the system.

In [29], the authors first found hotspots from large-scale real-world GPS data set by an improved DBSCAN algorithm. Then, a similar start-end point-based trajectory method was described to find out the potential trajectories between hotspots together with the density-based ε distance trajectory clustering algorithm in order to refine trajectory clusters. A weighted tree based route evaluation was derived, which included factors such as driving time, velocity, and endpoint attractiveness for optimal route evaluation. A case study was conducted to show the effectiveness of the method.

2.2.3 Mining the Route

Dijkstra's algorithm was a well-known algorithm [30] to find the shortest distance path between two nodes. The basic idea of Dijkstra's algorithm was to consider the corresponding cost of each neighboring node from the start node and choose the one with the least cost to expand. The process was repeated recursively until the destination node was reached. The other classical algorithm, A* algorithm, was proposed by Hart, et al. [31], which was a greedy best-first search algorithm. A* algorithm used a heuristic function to search from the start location towards the destination location. The heuristic function was divided into two functions: the path-cost function and the heuristic estimate function. The path-cost function was the real distance from start location to an intermediate location under examination, and the heuristic estimate function was the estimation distance from examined intermediate location to the destination location. One may notice that there might be a failure to find the shortest path when the algorithm cannot find the destination location. This could happen especially when reaching a node which had no available new best nodes for further expansion.

Lim & Kim [32] proposed a link-based shortest path algorithm to generate dissimilar paths to effectively reflect all turns in a real road network. The authors observed that heavy traffic may occur on some specific paths, which may lead to oversaturation of traffic. Therefore, they used an overlap degree function to exclude the overlapping alternatives. However, due to the dynamic traffic of the road network, the shortest path is not necessarily the fastest one.

The above papers studied how to mine the shortest recommended route for taxis. There are also other researchers that focus on how to provide the fastest recommended route so as to find passengers quickly. The fastest route problem is an extension of the shortest route recommendation problem. It changes the edge of the road network from the travel distance to the travel time.

Awasthi, et al. [33] proposed a rule-based method for evaluating the fastest paths in urban cities. In order to derive the predictive fastest path, a traffic log was used in this paper to build the statistical model. However, the model did not consider the starting time. In other words, the fastest paths may change at different starting times, and this model could provide unprofitable routes to taxi drivers.

Yuan, et al. [34] presented a method to calculate the fastest route to a destination at a given departure time. It minimized the expected distance to find a passenger. However, they did not consider the balance of taxis and passengers and do not report any experimental results in the real dataset.

An adaptive fastest path algorithm whose routing decisions are based on speed patterns mined from historical data was recently developed by Gonzalez, et al. [35]. They aimed to offer recommended routes that not only provides a set of driving conditions in real time but also reflects taxi driver preferences. Their research considered the fact that existing algorithms may send a driver through high crime areas of a city at night or through unsafe roads in order to save a few minutes of travel time. To avoid that situation, a road network partitioning algorithm was introduced. The algorithm used the hierarchy of roads to segment the network into areas and limits so the route search strategy can path segments that are actually frequently safely traveled based on a historical dataset. However, the algorithm cannot provide the route in real time.

Besides, a method was proposed in [36] to estimate travel times along the routes and predict arrival times at the nodes of a stochastic and dynamic network in real time. Their travel time estimator was developed based on a predictor-corrector form of the Kalman filter. It used the real-time and historical data of travel times along an arc of a transportation network to estimate the travel times along that arc for future times [36]. The predicted travel times were then used to estimate the arrival times at the nodes of the network. It was shown that, under fairly mild conditions, the developed travel and arrival time estimators are unbiased and that the error variance of the arrival time estimator is bounded. Simulation results are used to demonstrate the efficiency of the proposed algorithm.

2.3 Big trajectory data problem

Statistics revealed that the time complexity of existing recommendation methods is usually exponential [37]. Consequently, the issue of how to extract useful information from huge GPS trajectory in real-time becomes a very crucial problem. Moreover, the volume of trajectory data is large due to the high amount of taxis, and traditional trajectory mining methods are based on a centralized technology and can only run on one machine. Such high volumes of trajectory data could exceed the computational capacity of one machine. Therefore, the issue of how to load such huge amounts of trajectory data becomes another important problem. To solve this problem, work has been done on the spatial-temporal data structures as well as parallel processing based on GPS trajectories.

2.3.1 Spatial-Temporal Data Structure

Numerous spatial-temporal indexes have been proposed for speeding up searching efficiency. Among spatial access methods, one of the most popular structures is the R-tree. R-trees were developed for static points and regions. Dynamic data can be indexed by multiple R-trees, each capturing a different version of the data. There are also other data structures for specialized spatial-temporal dataset such as the Spatial-Temporal R-tree (STR-tree) and Trajectory-Bundle tree (TB-tree) [38]. An STR– tree is an R-tree based access method that considers the trajectory identity in the index. It organizes line segments of a trajectory according to both their spatial properties and the trajectories they belong to. And the TB-tree is a hybrid structure, which not only preserves trajectories but also allows R-tree to perform typical range search in the data. If trajectories are projected to the time-of-day domain, STR-tree index values from the projected trajectories could be used as an alternative representation of trajectories. Even through both above approaches would reduce the size of the problem of mining trajectories, it would not get the query results in real time.

Another classical data structure kd-tree is proposed to accelerate the K Nearest Neighbor search. The kd-trees are related to binary trees. A reference point is taken along with a reference dimension. Every other point that falls below the reference point for this dimension branches to the left, and all points with higher values branch to the right. However, kd-tree also has a disadvantage – its structure is sensitive to the order in which the data points are inserted, and the space partition can become very busy in certain areas. The result of this is that when searching for the nearest neighbours, more nodes have to be examined, and the computing time will increase.

Trestian, et al. [39] proposed a new approach to analyze human mobility. Their research focused on how to find the closest points of interest around user's position efficiently. In their method, an orthogonal kd-tree is used (see Figure 2-2) for its simplicity and performance for exacting k nearest-neighbors (i.e. k points of interest). Unlike kd-trees, orthogonal kd-tree only split space along the axis instead of partitioning a space in two sub-spaces along a hyperplane, i.e. a plane in three dimensional or a line in two-dimensional. In order to efficiently find the k nearest-neighbors of a location, orthogonal kd-tree first performed a depth-first search to reach the leaf node which corresponding to the space containing the desired location. Then, the recursion was performed to refine the k nearest-neighbors by searching closer points of interest in nearby spaces. Finally they evaluated user interaction with the points of interest by using the orthogonal kd-tree. However, the data structure also has the same drawbacks as the classic kd-tree mentioned above.

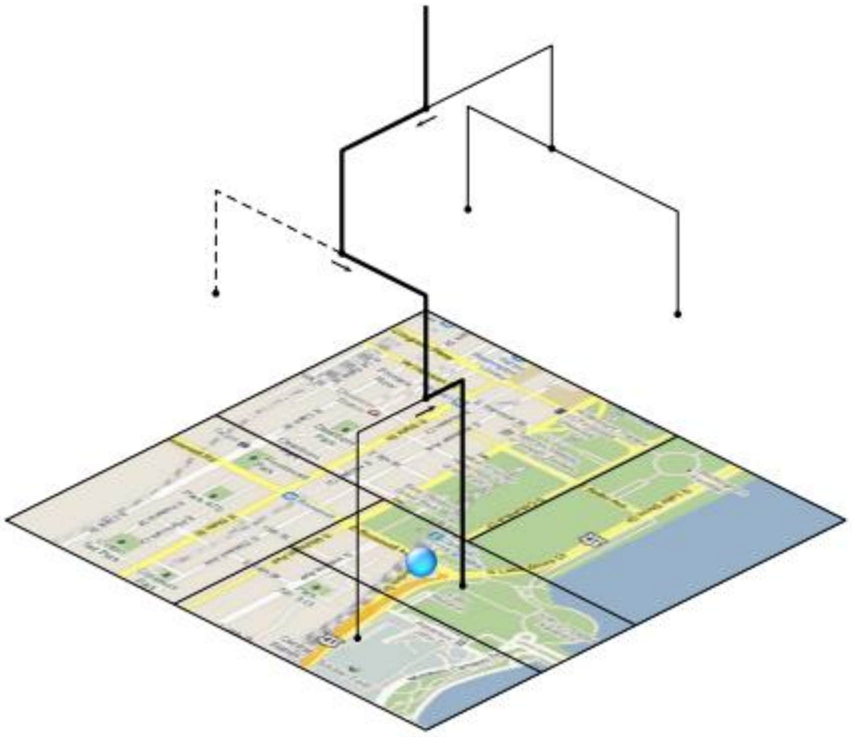


Figure 2-2. An orthogonal kd-tree [39]

Vlachos, et al. [40] presented a way to retrieve trajectories in the presence of noise effectively. They defined similarity functions, which were based on the longest sharable subsequence, which facilitates an intuitive notion of the similarity between trajectories. They used a distance measure to compute the similarity between trajectories of moving objects. Moreover, an efficient approximate algorithm that computed these distance measures, which was based on hierarchical clustering, was provided for similarity (nearest neighbor) queries. This algorithm partitioned trajectory data into sets according to the length first. Then, the index structure was applied to each set in order to search for the nearest trajectories. The experimental results of the index evaluation showed good speed-ups when applying the search for similar trajectories.

2.3.2 Parallel Computing

Facing huge volumes and complicated formatting problems of trajectory data, parallel computing will provide a promising new approach for such a vast amount of data. There are many research efforts on large-scale distributed storage and computing underway, such as GFS [41], MapReduce [42], and Map-reduce-merge [43], Dryad [44]. The efficiency of parallel computing has been shown by the performance of many data intensive applications. Recently, the framework named MapReduce has gained the world's attentions and has shown satisfactory performance in many data applications. It provides a framework that easily enables the processing of massive amounts of data over large clusters.

The MapReduce [45] [46] framework has a single master job machine and multiple task machines. The master machine manages the partitioning of input data, scheduling of tasks, machine failures, reassignment of failed tasks, inter-machine communications, and the monitoring of the task status. The task machines execute the tasks assigned by the master. Both input and output are stored in the file-system. MapReduce [45] [46] is best suited to deal with large datasets and is, therefore, ideal for mining large datasets that do not fit into physical memory. Figure 2.3 depicts the general MapReduce working mechanism.

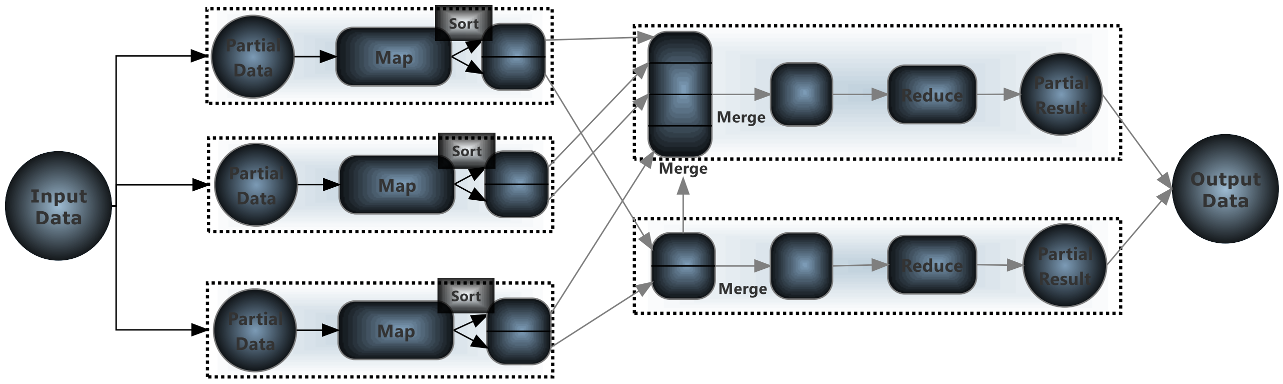


Figure 2-3. MapReduce Working Mechanism

A spatial temporal index algorithm named Layer-by-Layer Index Based on Grid Partition (LIBGP) which runs on MapReduce is proposed by Sun, et al. [47]. LIBGP may avoid problems of low trajectory recognition rates based on a grid. Also, it can retrieve gigabytes of data efficiently and shows good scalability in terms of data dimension and size. Ma, et al. [48] presented a paradigm called PRADASE (Query Processing of Massive trajectory data based on MapReduce) for query processing over massive trajectory data based on MapReduce, and it also relies on GFS (Google File System) storage, which partitions large amounts of historical trajectory data sets into data chunks.

These data chunks are distributed into different nodes, thus, making the whole system more available and reliable. The experiments showed this append-only scheme of the MapReduce storage model may be an adequate base for handling the updates of moving objects.

Also, querying kNN from large-scale spatial data by using the MapReduce framework has been studied intensely. Fang, et al. [49] presented an efficient framework for answering k-NN join queries using MapReduce. It partitioned the trajectories using spatial grids first, and then a new tight TDB (Time-Dependent Upper Pound) was proposed to improve their efficiency, which can achieve a higher efficiency by pruning trajectories in parallel with a tighter upper bound. Eldawy & Mokbel [50] presented Spatial Hadoop as the first fully-fledged MapReduce framework for spatial data, which employs a simple spatial high level language, a two-level spatial index structure, basic spatial components built inside the MapReduce layer, and three basic spatial operations: range queries, kNN queries, and spatial join. Spatial Hadoop can achieve an order of magnitude of better performance than Hadoop for spatial data. Also, it is open source and allows researchers to further extend its functionality. (Li, et al. [51] used MapReduce to analyze massive bus trajectory data. The study focused on correcting the travelling direction based on the GPS trajectory and projecting the GPS readings to the corresponding road network. By utilizing the geometric and topological information of the road network, the exact projection point from bus GPS readings can be obtained, and the travelling direction of the bus in each trip may also be determined. To improve the algorithm, the kNN algorithm is used to evaluate the final results of the travelling direction, and the Map-Reduce framework is used to improve the computing efficiency.

The experiments showed that the accuracy of the direction was significantly improved when employing the kNN algorithm, and the computational efficiency was also improved by at least 100 times after utilizing parallel processing framework.

However, none of above methods integrated data structure into parallel computing model. Although they can process big trajectory data problem due to their great data organization capacity, the properties of spatio-temporal data have not be fully used. By considering “data structure + parallel model” data processing architecture, the spatial computation efficiency can be significantly improved by utilizing data structure to partition space according to GPS data’s distribution information and also by using parallel computing model to provide query paralleled operation.

CHAPTER THREE: PROFITABLE ROUTE RECOMMENDER

This chapter presents a personalized route recommendation algorithm for a given group of taxis. The formal problem statement of the thesis is introduced in Section 3.1. Section 3.2 describes how to build a temporal probabilistic grid network. In Section 3.3, the framework of recommendation method is presented. Two algorithms, named shortest expected cruising route (SECR) and adaptive shortest expected cruising route (ASECR) algorithm, are developed to generate potential profitable route for each taxi.

3.1 Problem Statement

In this subsection, four concepts are formalized to specify the problem of how to recommend a profitable route for taxi, including ***taxi***, ***state***, ***GPS-reading*** and ***trip***.

Definition 1: A ***taxi*** t is defined as a 5-tuple, denoted as:

$$t = (g_o, g_d, r, \tau, s),$$

where g_o and g_d are the origin and destination of the taxi. r is a route from the origin to the destination. τ is the starting time of the route r. s is a Boolean value to show whether the taxi is occupied.

GPS-readings describe the trajectories of taxis, recording their location, time and state at predefined time interval.

Definition 2: A ***GPS-reading*** $\gamma^t$of a taxi t is defined as a 3-tuple, denoted as:

$$\gamma^t = (\tau, l, s),$$

Where τ is the time stamp of the GPS-reading. *l* is the location of the taxi at time τ and s is a Boolean value showing whether the taxi was occupied by a passenger at time τ .

Definition 3: The ***state*** of a taxi t is determined using its GPS-trajectory:

$$state(t) = \begin{cases} cruising, if\ \gamma_i^t.s = false \\ occupied, if\ \gamma_i^t.s = true \\ out-of-\ service, if\ \gamma_i^t.\tau - \gamma_{i-1}^t.\tau > \theta \end{cases},$$

where s is the Boolean value of the GPS-reading and θ is the threshold to determine whether the taxi is in service. A taxi is in *Cruising* state when the taxi is not occupied by a passage, i.e., s is false. If a passenger is on the taxi, i.e. s is true, the taxi is in the *occupied* state. When the time difference between the two GPS readings are greater than the threshold θ, then the taxi is in *Out-of-service* state. Figure 3-1 shows three-state state diagram. Unlike *Cruising* and *Occupied* states, the *out-of-service* state is not explicitly stated in the GPS-readings. When a taxi goes *out of service*, its GPS-trajectory is not updated for a long time. The *out-of-service* can be detected by using a threshold on the updating time of the GPS-readings. And this parameter θ is an empirical value and is specified manually in implementation. When the taxi was *out-of-service*, these two GPS-readings are not "logically" consecutive. It means that their physical distance is not a good estimation of the distance that taxi travelled in that time interval. Therefore, the *out-of-service* state is used to divide trajectories into sub-trajectories, or called "trips" (as shown in Figure 3-1).

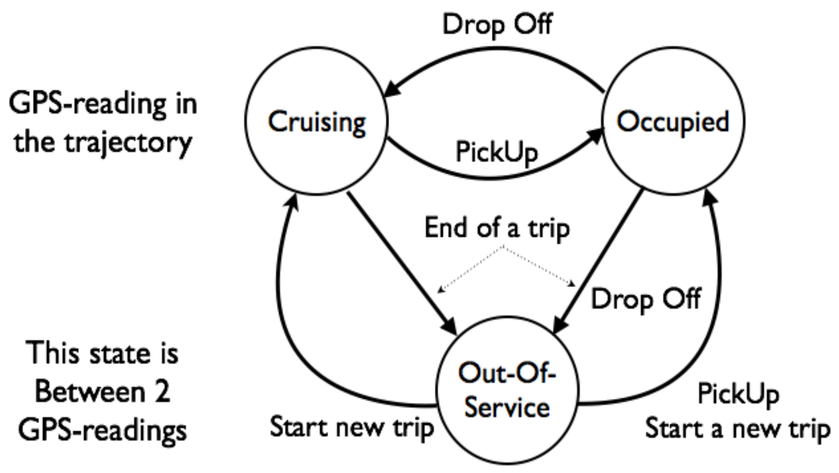


Figure 3-1. Three–State Diagram

Definition 4: A ***trip*** of a taxi t is defined as a set of GPS readings, denoted as:

$$trip^t = (\gamma_1^t, \gamma_2^t, \gamma_3^t, \ldots, \gamma_m^t); \text{where } 0 < \gamma_i^t.\tau - \gamma_{i-1}^t.\tau \leq \theta, 1 < i \leq m.$$

The GPS-trajectory of taxi *t* can be defined as a set of trips. Each trip is a sub-trajectory of the taxi during which the taxi *t* is in service. So the trips can be determined by processing the whole GPS-trajectory.

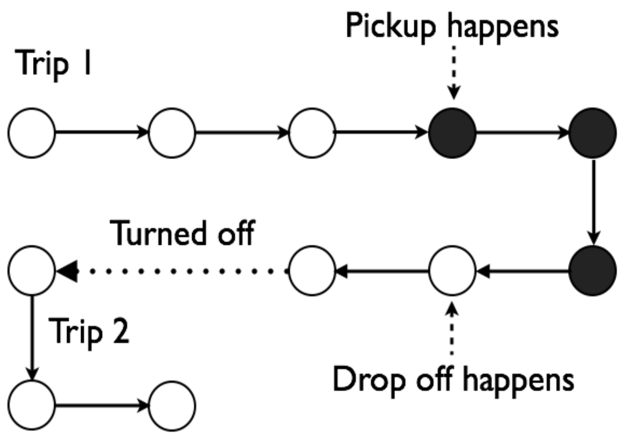


Figure 3-2. A Trajectory Divided into 2 Trips (Circles represent GPS-reading. Filled circles represent occupied state).

Since GPS-readings from taxis are generally imprecise, directly matching the GPS-readings on existing road network could be problematic. In addition, the operations on the existing road network are computational expensive due to the large amount of vertices (i.e. junctions) and edges (i.e. road segments) on the road network. Taking Figure 3-3 as an example, it is easily to observe that points 'c' and 'f' might not be mapped to the actual locations. In addition, the operations on the existing road network are computational expensive due to the large amount number of vertices and edges on real road network. In this thesis, a temporal probabilistic grid network from historical GPS trajectories is defined to divide the interested region into small grids and summarize the temporal and spatial information from taxis' GPS trajectories.

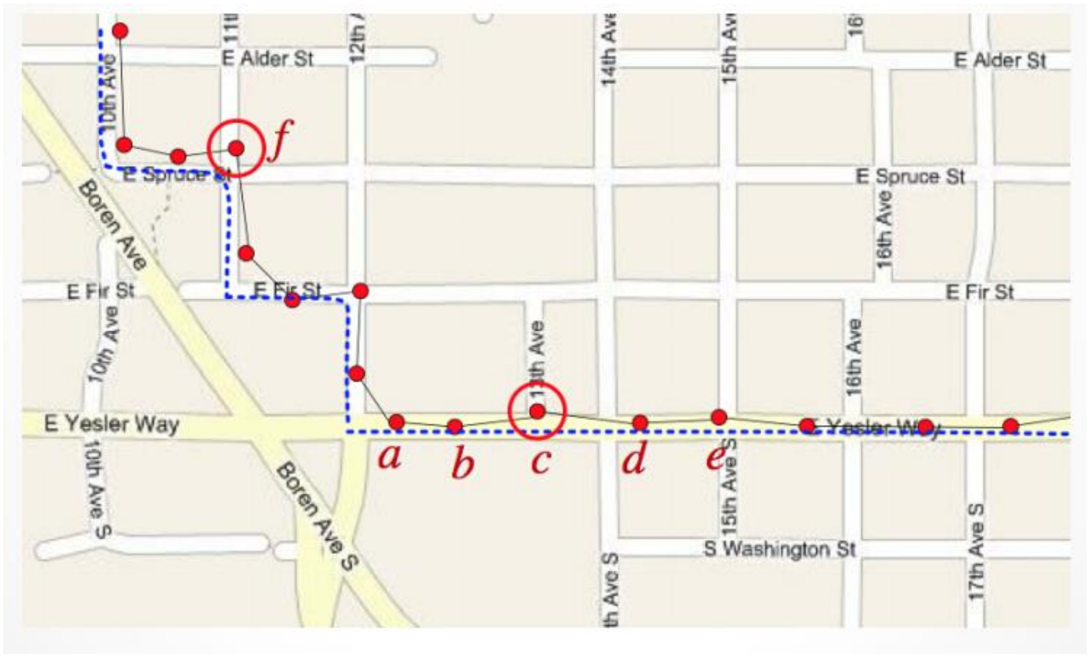


Figure 3-3. An Example of Imprecise GPS-readings

Definition 5: A ***grid*** g is a 4-tuple, denoted as:

$$g = (l, p, c, \tau),$$

where l is the center location of the grid g, p and c are the temporal probability and capacity of g at time τ. The temporal probability is the probability that a taxi finds a passenger in the grid g at the time τ. The capacity is the number of the possible passengers of the grid g at the time τ.

Definition 6: A ***grid network*** GN is a weighted directed graph, denoted as:

$$GN = (G, E, D),$$

$$\text{where} \quad \begin{aligned} G &= \{g_1,\ g_2, \ldots, g_n\}, \\ E &= \{e_{ij} \mid g_i, g_j \in G\}, \\ D &= \{d_{ij} \mid e_{ij} \in E\}, \end{aligned}$$

where G is a set of grids, and E is a set of edges connecting grids. An edge is built to connect two neighbor grids if consecutive GPS-readings exist between the two grids. D is a set of weights of the edges, representing the physical traveling distances of edges.

To measure the performance of a taxi, a distance performance is defined.

Definition 7: Given a set of taxis T and the time interval $[\tau 1, \tau 2]$, the taxis' ***distance performance*** starting between $[\tau 1, \tau 2]$ is defined as:

$$P(T) = \frac{\sum_{t \epsilon T} \sum_{t.r} dist_{occupied}}{\sum_{t \epsilon T} \sum_{t.r} (dist_{occupied} + dist_{cruising})}$$

where $dist_{cruising}$is the travelled distance that a taxi is in cruising state. $dist_{occupied}$ is the travelled distance that a taxi is in occupied state.

The distance performance measures the ratio of distance travelled by all the taxis in T having a passenger on board to the whole distance they travelled. A larger value of distance performance indicates the higher distance efficiency of the set of taxis. In this thesis, assume the distance performance reflects the profits of taxis.

Hence, the problem of finding profitable routes for a set of taxis can be defined as:

Definition 8: Given a set of taxis T, the taxi ***profitable routing problem*** is to recommend routes to a set of taxis so that the overall distance performance of taxis is maximized. It is formulized as:

$$Arg_{t.r} \max \mathrm{P}(T) \ ,$$

where $t \epsilon T$. This definition is to recommend routes to a set of taxis so that the overall profit of all taxis is maximized. This can be achieved by minimizing the cruising distances for all taxis. In other words, the goal is to maximize the average distance performance of the whole group of the taxis.

3.2 Building Temporal Probabilistic Grid Network

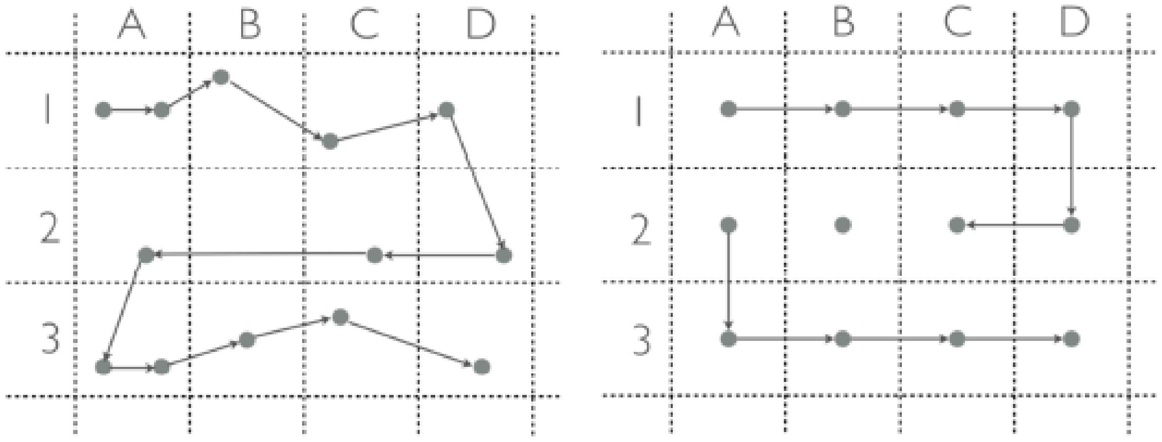
Since GPS-readings are generally imprecise, one reading may relate to the wrong road segment. Directly matching the GPS-readings on existing road network could be problematic. Besides, the size of GPS-readings is very large which will be computationally expensive. To avoid those problems, a temporal probabilistic grid network is carrying out. This subsection presents the method to build a temporal probabilistic grid network from taxi trajectories, which includes two steps: grid network generation and grid’s probability and capability calculation.

3.2.1 Generation of Grid Network

First, the primary grid network is generated from GPS trajectory. The interested region will be divided into small grids and GPS-readings will be projected onto grids according to their coordinates. The following two steps briefly show how to build the grid network:

First, the GPS-readings are overlaid with gridlines, showing in which grid each of the GPS-readings is placed (shown in Figure 3-4 (a)).

Second, the center of each grid becomes a node of the network, called grid node. An edge exists between two grid nodes if a pair of consecutive GPS-readings satisfies the two conditions: 1) Two readings are placed in different grids; 2) The two grids are neighbors. Figure 3-4 (b) shows the grid network derived from the GPS-readings.



(a) **(b)**

Figure 3-4. Deriving the Road Grid Network from Trajectories

The size of grids is important to build the grid network. If the grid size is too big, the grids cannot be considered as a point since there would be more than one GPS-reading located in the same grid and it will make the graph less accurate for losing the capacity of tracking taxi's every movement from the GPS data sequences. If it is too small, then the distance between most of the consecutive GPS readings is larger than the area size. It is impossible to build a connected graph processing the GPS-trajectories because no pair of consecutive GPS-readings will be placed in the neighboring grids.

Before determine the grid size, the first needed is identifying the *out-of-service* states from each trajectory and break the trajectory into trips. Since the distance between the two consecutive readings is not meaningful when the taxi is *out-of-service*.

The *out-of-service* state is determined by analyzing the trajectories to see how frequent the GPS-readings are reported. To do this the time interval between every two consecutive GPS-readings is considered in each trajectory. The following empirical distribution function (ECDF) is used to find the threshold for out-of-service state, which is denoted by $P(x_i \leq \omega)$. The empirical distribution of time intervals between two consecutive GPS-readings can be expressed as:

$$P(x_i \leq \omega) = \frac{|x_i \mid x_i \leq \omega\,|}{n}; where\ x_i = \gamma_i.\tau - \gamma_{i-1}.\tau\,,$$

where n is the total number of the time intervals of the GPS-readings and x_i is the time interval of two consecutive GPS-readings. $P(x_i \leq \omega)$ is the ratio of the number of time intervals between two consecutive GPS-readings less than ω and the total number of the intervals of GPS-readings.

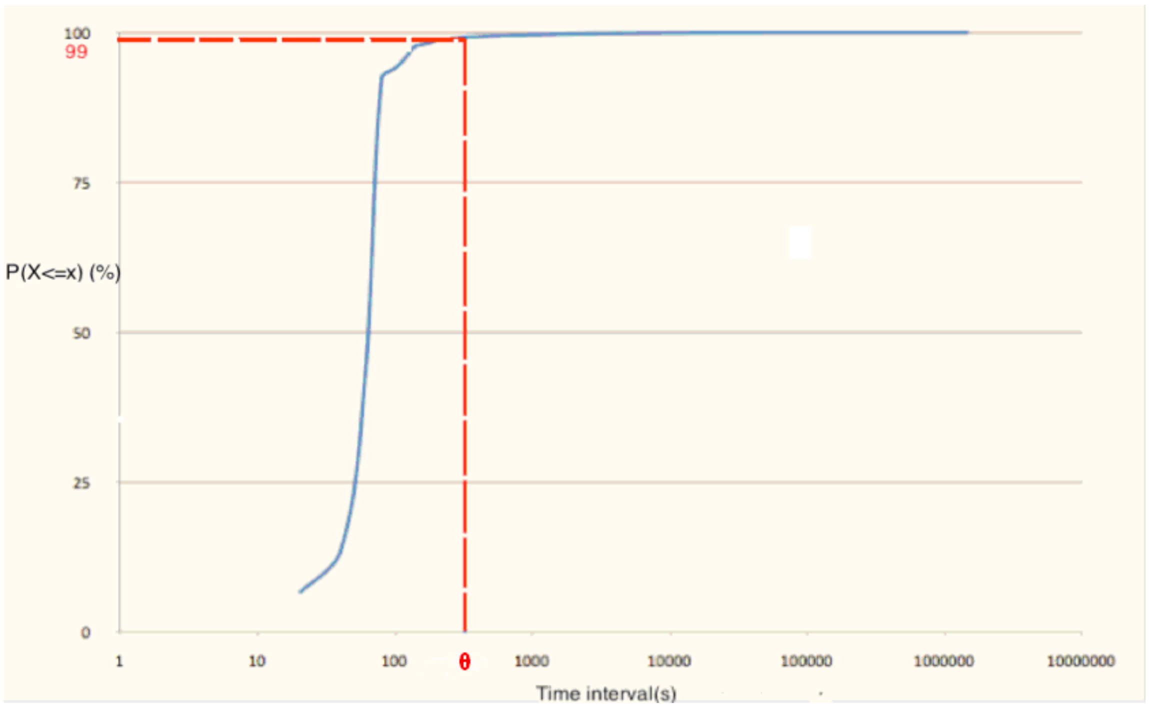


Figure 3-5. Distribution of the Time Interval between Consecutive GPS-Readings

Figure 3-5 shows an example plot from our San Francisco experimental dataset. From the figure, about 99% of the consecutive GPS-readings have the time interval of less than 420 seconds. This means most time interval between two consecutive GPS is less than 7 minutes. If the time interval is more than 7 minutes, then assume that taxi is out-of-service between those two GPS-readings. So 7 minutes can be used as the threshold for *out-of-services* in this case. After determining the *out-of-services* threshold, the GPS trajectories can be divided into *trips*.

After identifying the time interval threshold for out-of-service consecutive GPS-readings, the similar method can be used to plot the distances between two consecutive GPS-readings so as to determine the grid size. The distance that is greater than the distances between most of the two consecutive GPS-readings will be selected as the grid size. The empirical distribution of distances between two consecutive GPS-readings can be formulized as:

$$P(x_i \leq \sigma) = \frac{|x_i \mid x_i \leq \sigma\,|}{n}; where\ x_i = dist(\gamma_{i-1}.l - \gamma_i.l)\ ,$$

where $P(x_i \leq \sigma)$ is the ratio of the number of distance intervals between two consecutive GPS-readings less than σ and the total number of the intervals of GPS-readings. $dist(\gamma_{i-1}.l - \gamma_i.l)$ is the Euclidean distance between two consecutive GPS-readings. n is the total number of the distance intervals of the GPS-readings and x_i is the distance interval of two consecutive GPS-readings.

Figure 3-6 shows a plot of the distribution of distances between consecutive GPS-readings. It shows that the gradient of the distribution curve approaches to zero at around 550 meters, which represents the consecutive GPS-readings are at most 550 meters apart. Then 550 meters can be a good value for the grid size.

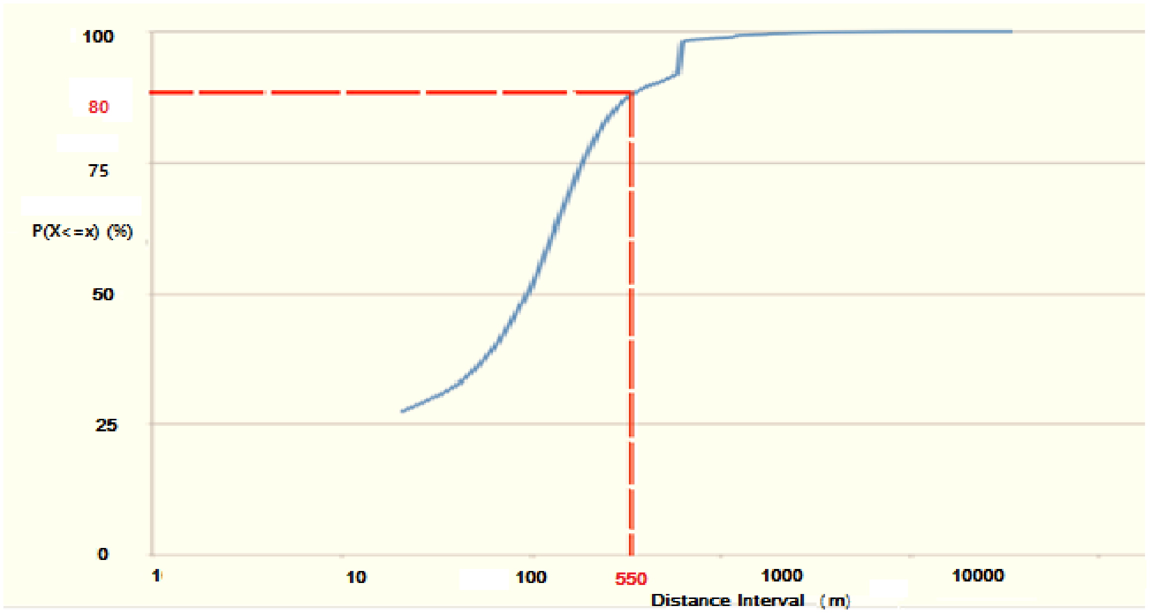


Figure 3-6. Distribution of the Distance between Consecutive GPS-Readings

3.2.2 Calculation of Probability and Capacity of Grids

Probability of finding a passenger in a location and capacity of a grid are two temporal properties of the grid. After generating the primary grid network, the two temporal properties will be calculated.

The probability is the expected–value of passenger's existence in the grid, which is how likely for a taxi to find a passenger. For each recommended pick-up location (the centroid of historical potential profitable grid), the probability of a pick-up event can be computed based on historical GPS-trajectory data. The capacity, on the other hand, is the maximum number of taxis that can enter the grid without affecting the passenger–taxi balance in the grid. After carefully observing historical GPS-readings of each grid, it is easily to notice that there are relative more pick-up events passengers in some places than others. In other words, more taxis can be recommended into the same grid within the same time interval. The definitions of the two properties are defined as follows.

Definition 9: The probability of finding a passenger in a grid l in a given time interval $[\tau 1, \tau 2]$ is defined as:

$$p(l, \tau 1, \tau 2) = \frac{|P|}{|E|},$$

$$where \ \ P = \{t_i \,|t_i.g_i = l,\ \ t_i.s = occupied,\ \ t_{i-1}.s \neq occupied, \tau_1 \ \leq \ t_i.\tau \ \leq \ \tau_2\}$$
$$E = \{t_i \,|t_i.g_i = l,\ \ t_{i-1}.g_i \neq l, \tau_1 \ \leq \ t_i.\tau \ \leq \ \tau_2\}$$

where $|P|$ is the number of taxis showing pickups in grid l in the time interval $[\tau 1, \tau 2]$ and $|E|$ is the total number of taxis entering grid l in the time interval$[\tau 1, \tau 2]$.

However, if all taxis are assigned to the same set of grids, it may result in non-equilibrium of supply and demand. To avoid that, the capacity of a grid is defined to control the number of taxis assigned to it.

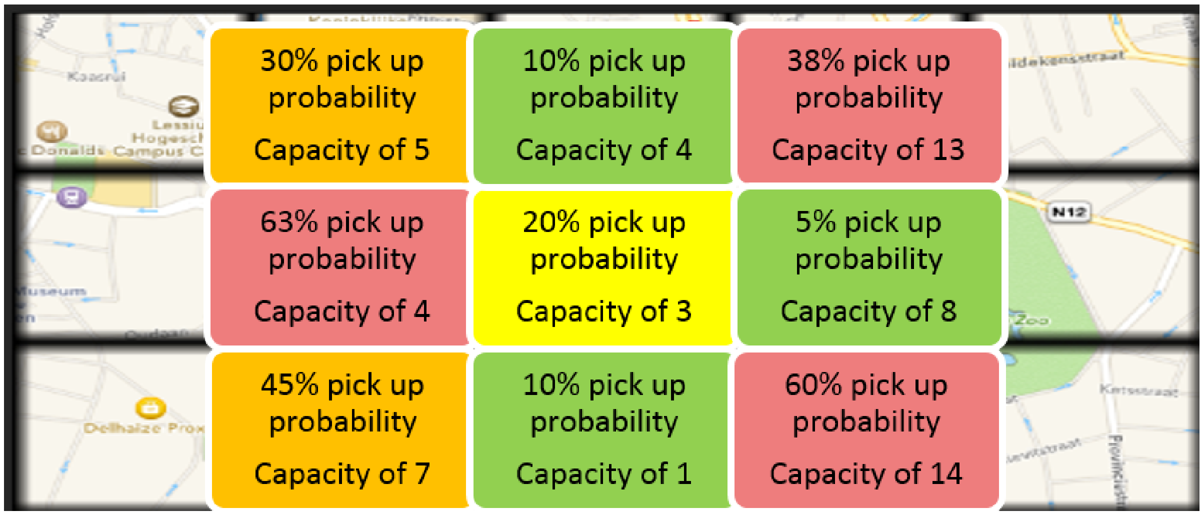

30% pick up probability Capacity of 5
10% pick up probability Capacity of 4
38% pick up probability Capacity of 13
63% pick up probability Capacity of 4
20% pick up probability Capacity of 3
5% pick up probability Capacity of 8
45% pick up probability Capacity of 7
10% pick up probability Capacity of 1
60% pick up probability Capacity of 14

Figure 3-7. An Illustration of Temporal Features of Grid Network

Definition 10: The capacity of a grid l in a time interval $[\tau_1, \tau_2]$ is defined as:

$$c(l, \tau) = c(l, \tau_1, \tau_2) = |E|$$

where $|E|$shows the taxis picking up a passenger at the grid l . In other words, this parameter implies the number of pickups of the grid in each recommendation in a given time interval.

As shown in Figure 3-7, the map is divided into small sub-rectangle, each rectangle represent one grid. For each grid, the probability and capacity is calculated in advance.

3.3 Profitable Route Recommender

The framework of the profitable route recommendation algorithm is shown in Figure 3-8. To reduce the total cruising distance and balance the workload of a group of empty taxis, first potential profitable grids are assigned to each taxi with respect to the probability and capacity of the grids in line 1. Then the potential profitable route is recommended for each taxi. In line 3, the recommended route connects assigned potential profitable grids with potential minimum cruising distance. After finding a passenger, taxis will release their assigned potential profitable grids. Thus, as shown in line 4, the capacity of each released potential profitable grid will be updated. In the following, each step will be discussed in detail.

```
Algorithm recommendProfitableRoutes (T, G)
Input: T: A set of empty taxis; G: A grid network.
1.  ProfitableRouteRecommender.assignGridsToTaxis (T, G);
2.  for each taxi t in T {
3.        ProfitableRouteRecommender.findShortestExpectedCruisingRoute (t);
4.        ProfitableRouteRecommender.updateCapacity (t, G);
5. }
```

Figure 3-8. recommendProfitableRoutes Algorithm

3.3.1 Potential Profitable Grids Assignment

As discussed in Section 3.2, each grid of the grid network has the probability and capability. The potential profitable grids assignment aims to assign a set of grids which have the probability to pick up passengers to each taxi. Each potential profitable grid can only be assigned to the limited number of taxis based on its capacity. In the case that a taxi does not have potential profitable grids assigned, the algorithm gets one potential profitable grid from the nearest rich taxi and assigns it to this taxi. A rich taxi is a taxi that has been assigned to multiple grids.

```
Algorithm assignGridsToTaxis (T, G)

Input: T: A set of empty taxis; G: A grid network.

1.   for each grid g in G {
2.           if g.getCapacity( ) > 0 {
3.                capacity = g.getCapacity( );
4.                g.assignedTaxis = g.findNearestTaxis ( T, capacity);
5.                for each taxi t in g.assignedTaxis
6.                      t.addProfitableGrids(g);
7.                g.setCapacity(g.getCapacity( ) – g.countAssignedTaxis( ));
8.            }
9    }
10.   for each taxi t in T  {
11.           if t.getProfitableGrids( ) is empty
12.                StarvingTaxis.add(t);
13.  }
14.  for each taxt t in StarvingTaxis {
15.           richTaxi = t.getNearestRichTaxi ( );
16.           assignedProfitableGrids = richTaxi.getprofitableGrids( );
17.           selectedGrid = t.findNearestGrid (assignedProfitableGrids);
18.            t.addProfitableGrid(selectedGrid);
19.            richTaxi.removeGrid(selectedGrid);
20    }
```

Figure 3-9. assignGridsToTaxis Algorithm

As shown in Figure 3-9. The algorithm examines all potential profitable grids that have higher capacity than the number of taxis in them. If the capacity of the grid is c, this algorithm assigns the grid to the c-nearest taxis (Lines 1 to 8). Then in Lines 10 to 13, all taxis will be checked to see whether it is a starving taxi, i.e., a taxi is not assigned to any profitable grid. For each starving taxis, the algorithm selects the nearest potential grid from the rich taxi, and assigns that the potential profitable grid to the starving taxi (Lines 14 to 20). By doing that, the starving taxi can be brought into the area with more potential passengers.

3.3.2 Profitable Route Recommendation Algorithms

In this section, a route with the shortest expected cruising distance that connects the assigned potential profitable grids of each taxi is proposed. First, the concept of the expected cruising distance is introduced. Then two methods are proposed to search the route with the shortest expected cruising distance route.

3.3.2.1 Expected Cruising Distance

As discussed, the approach to find a profitable route for each taxi is to find the passengers with the shortest expected cruising distance. This section will discuss how to calculate the expected cruising distance from the taxi to a potential passenger based on the grids' probabilities.

If a taxi is going to move along a route $r = \{l_1, l_2, \dots, l_n\}$ in time interval $[\tau 1, \tau 2]$ from grid l_0, it may pick up passengers at any grid of the route. For each grid $l_i (1 \leq i \leq n)$, the probability that the taxi will pick up a passenger at the grid can be calculated with the following equation:

$$p(l_i|r,\tau1,\tau2) = \begin{cases} p(l_i,\tau1,\tau2), & i = 1 \\ p(l_i,\tau1,\tau2)\prod_{j=1}^{i-1}(1 - p(l_i,\tau1,\tau2)\,), & i > 1 \end{cases},$$

where $p(l_i|r,\tau1,\tau2)$ is a conditional probability that taxi does not pick up a passenger at previous grids but in grid l_i.

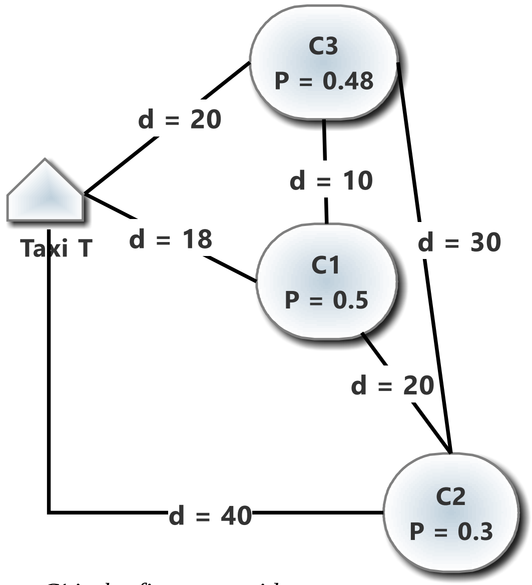
Note that the cruising distance before picking up a passenger at l_i is$\sum_{j=0}^{i-1} dist(l_j,l_{j+1})$, while $dist(l_j,l_{j+1})$ is the Euclidean distance between grid l_j and gridl_{j+1}. The potential expected cruising distance could be calculated by integrating the distribution of the conditional probability with the distribution of the cruising distance before picking a passenger. So the expected cruising distance can be defined as follows:

Definition 11: Given a taxi t located at grid l_0, and a set of potential profitable grids with a route sequence $r = \{l_1, l_2, \ldots, l_n\}$, the ***expected cruising distance*** in the time interval $[\tau1,\tau2]$ is defined as:

$$dist_{expected}(t.r) = \sum_{i=1}^{n}\left(p(l_i|r,\tau1,\tau2)\sum_{j=0}^{i-1} dist(l_j,l_{j+1})\right),$$

Definition 12: Given a taxi t and a set of potential profitable grids $G = \{l_1, l_2, \ldots, l_n\}$, ***the shortest expected cruising distance*** in the time interval$[\tau1,\tau2]$ is the minimal expected cruising distance of all possible routes which connect all grids in G.

Figure 3-10 is an illustration example. For taxi T, C1, C2 and C3 are its assigned potential profitable grids. As shown in the figure, if C1 is assigned as the first stop, the shortest expected cruising distance is 22.8, and corresponding route is C1-C2-C3. The shortest expected cruising distance is 26.8 for the route C2-C1-C3 if C2 is assigned as the first stop. For C3 as the first stop, the shortest expected cruising distance is 20.6 and the corresponding route is C3-C1-C2. So in this case, the shortest expected distance of recommended route is the one with C3 as the first stop, which is $C3 - C1 - C2$. Meanwhile, we notice that though grid C1 is the closest pick-up location for requested taxi, the algorithm will not recommend C1 as the first stop for the recommended sequence



$C1\ is\ the\ first\ stop\ grid:$

$$dist_{expected}(t.r) = P(C1)dist(t,C1) + P(C2)dist(C1,C2) + P(C3)dist(C2,C3) = 22.8$$

$$dist_{expected}(t.r) = P(C1)dist(t,C1) + P(C3)dist(C1,C3) + P(C2)dist(C3,C2) = 29.4$$

$C2\ is\ the\ first\ stop\ grid:$

$$dist_{expected}(t.r) = P(C2)dist(t,C2) + P(C1)dist(C2,C1) + P(C3)dist(C1,C3) = 26.8$$

$$dist_{expected}(t.r) = P(C2)dist(t,C2) + P(C3)dist(C2,C3) + P(C1)dist(C3,C1) = 31.4$$

$C3\ is\ the\ first\ stop\ grid:$

$$dist_{expected}(t.r) = P(C3)dist(t,C3) + P(C1)dist(C3,C1) + P(C2)dist(C1,C2) = 20.6$$

$$dist_{expected}(t.r) = P(C3)dist(t,C3) + P(C2)dist(C3,C2) + P(C1)dist(C2,C1) = 28.6$$

Figure 3-10. An Illustration Example

3.3.2.2 Shortest Expected Cruising Route (SECR) Algorithm

To find the shortest expected cruising distance, a naive solution is to check all possible combinations of the assigned profitable grids and find the shortest cruising routes among them. The complexity for this solution is O(n!), where n is the number of assigned grids. Since some taxis may have been assigned more grids than others, the complexity could be exponential. The SECR algorithm is proposed to simplify the problem by setting a threshold K on the number of grids in a recommended route, which not only reduce time cost in computation but also avoid the problem that most resources are assigned to very few taxis by avoiding recommending a route that contain too many potential profitable grids. Then searches profitable route by calculating the expected cruising distance of all the possible sequences in the K nearest profitable assigned grids. The one with the shortest expected distance will be selected as the recommended sequence based on the K grids.

The pseudo code of SECR algorithm is shown in Figure 3-11. It first finds the K nearest potential profitable grids for taxi and calls *generateCandidateRoutes* function to enumerate all the possible sequence from the K nearest potential profitable grids as the route candidates. Then the *findShortestExpectedCruisingDistanceRoute* algorithm is called in line 5 to find the shortest cruising distance from the route candidates. As illustrated in *findShortestExpectedCruisingDistanceRoute* algorithm, when encounters a route whose expected cruising distance is shorter than other route candidates, this route will be selected as the shortest expected cruising route.

```
Algorithm findShorestExpectedCruisingRoute(t)
Input: t is a taxi that needs route recommendation.
Output: recommendedRoute is the shortest expected cruising route.
1.  Initialize recommendedRoute to Null;
2.  currentGrid = t.getCurrentGrid( );
3.  assignGrids = t.getTopKNearestAssignedGrid(currentGrid);
4.  routeCandidates = t.generateCandidateRoutes(assignGrids);
5.  recommendedRoute = t.findShortestExpectedCruisingRoute(routeCandidates);
6.  return recommendedRoute;

findShortestExpectedCruisingRoute (candidateRoutes)
Input: candidateRoutes is all the possible sequence in assigned grids.
Output: recommendedRoute is the shortest cruising route.
1.      Initialize shortestExpectedCruisingDistance to INT_MAX;
2.      route = candidateRoutes.getFirst( );
3.      while route is not NULL{
4.              distance = calculateExpectedCruisingDistance(route);
5.              if shortestExpectedCruisingDistance > distance
6.                      shortestExpectedCruisingDistance = distance;
7.                      recommendedRoute = route;
8.              route = candidateRoutes.getNext( );}
9.      return recommendedRoute;
```

Figure 3-11. findShorestExpectedCruisingRoute Algorithm

3.3.2.3 Adaptive Shortest Expected Cruising Route (ASECR) Algorithm

The SECR algorithm selects the K-nearest potential profitable grids before finding the shortest expected cruising path. However as the time and location of a taxi keeps changing, the algorithm runs the risk of retrieving the out of date potential profitable grids to build the recommended route. So updating taxis' potential profitable grids based on the taxi's movement will provide better routes. So in this section, the ASECR algorithm will address the problem on finding the K nearest potential profitable grids at each grid along the recommended route. When the potential profitable grids for a taxi are updated, the ASECR algorithm will reassign the potential profitable grids and recalculate the shortest expected cruising route based on taxi's current location and current time.

```
Algorithm findAdaptiveShorestExpectedCruisingRoute (t, T, G)
Input: t is a taxi that needs route recommendation; T is a set of empty taxis; G is the grid network.
1.     while taxi t is empty  // when taxi arrives each grid{
2.         currentTime= t. getCurrentTime( );
3.         currentGrid = t. getCurrentGrid( );
4.         assignGridstoTaxis(T, G);
5.         assignGrids = t.getTopKNearestAssignedGrid(currentGrid);
6.         findShorestExpectedCruisingRoute(currentGrid, assignGrids);
7.     }
```

Figure 3-12. findAdaptiveShorestExpectedCruisingRoute Algorithm

Figure 3-12 shows the pseudo code of the ASECR algorithm. If a taxi cannot find a passenger, the algorithm loops and searches updated potential profitable grids for taxi by calling *assignGridstoTaxis* algorithm in Figure 3-10. After updating the assigned grids, line 6 finds the route with the shortest cruising distance for taxi. The loop continues until the taxi picks up a passenger.

CHAPTER FOUR: QUERY PERFORMANCE IMPROVEMENT

The ASECR Algorithm can produce a profitable route for each taxi based on the current time and the taxi's location. However, the algorithm needs to constantly update the K nearest potential profitable (kNN) grids, and the challenge of this type of operation is to efficiently perform frequent location updates. Therefore, this thesis investigates a new data structure KdS-tree to address this problem, which aims to improve k-NN query performance while reducing the update cost. Furthermore, the volume of the GPS dataset is huge and cannot be fully loaded into memory, which means the same data has to be recalled repeatedly and the disk needs to be accessed frequently. Since our algorithm is built based on the grid network which partitions the space into regular regions and is inherently suitable for parallel computing, a MapReduce model is explored in this thesis to get the response time within a reasonable time range.

This chapter focuses on the discussion of a MapReduce model, which consists of a series of MapReduce jobs, with the kdS-tree data structure to provide profitable routing recommendation. The chapter is organized as follows. Section 4.1 presents the detail of data structure for kdS –tree. Section 4.2 introduces MapReduce parallelization to our approach for ASECR Algorithm.

4.1 kNN Query based on KdS-tree

As mentioned, ASECR algorithm constantly updates the K nearest potential profitable grids based on current time and taxi's current location. Efficiently finding K nearest neighborhood (kNN) has been discussed in many literatures and can be solved through spatial index structures, such as kd-tree [52] and R-tree [53] [54]. However, these spatial indexes do not perform well for continuous kNN because they are not designed for frequently updating tree structures and efficiently querying profitable potential grids. In order to address this problem, a new data structure named kdS-tree is designed to solve the problem.

kdS-tree is extended from kd-tree and ball-tree [55]. It is a binary tree in which each node is associated with a circle. Similar as points are located on the boundaries of rectangles in kd-trees, the grid points locate on the circles in kdS-tree. Instead of only having one point on splitting line for 2-dimensional kd-tree, at least two points are form and on the splitting circles for kdS-tree, which reduce the depth of the tree. kdS-tree is constructed in a top-down manner. Given a set of profitable potential grids, first, a minimum enclosing circle is created to include all grids. Each node of kdS-tree contains five fields: center and radius of the associated circle, profitable grid nodes located on circle, left subtree and right subtree. The root node of a kdS-tree has the circle including all potential profitable grids. Then, two grid nodes are identified inside the circle as two seeds, between which is the largest distance among all pairs of grid nodes inside the circle. All other grid nodes inside the circle are assigned to the closer seed to form two clusters, and two minimum enclosing circles are formed for two clusters. Based on the x-coordinate of center of these two clusters, the two circles will be added to the left and right subtree. For each cluster, the procedure repeats until no profitable grid is inside the circle.

Each node in the kdS-tree is an enclosing circle with minimum radius to include all the assigned profitable grids and has the profitable grids that are right on the circle. As shown in Figure 4-1, Figure 4-1 (a) shows 21 profitable potential grids. Figure 4-1 (b) shows the generated minimum enclosing circles, and Figure 4-1 (c) shows resulting kdS-tree.

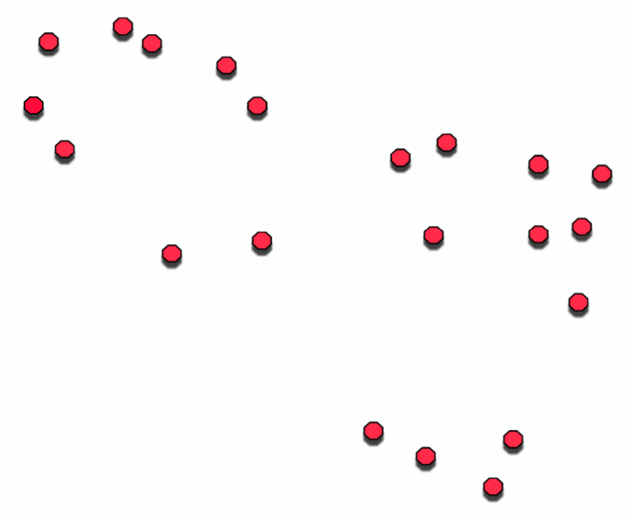


Figure 4-1 (a). kdS-tree for 21 Grids: the Profitable Grids

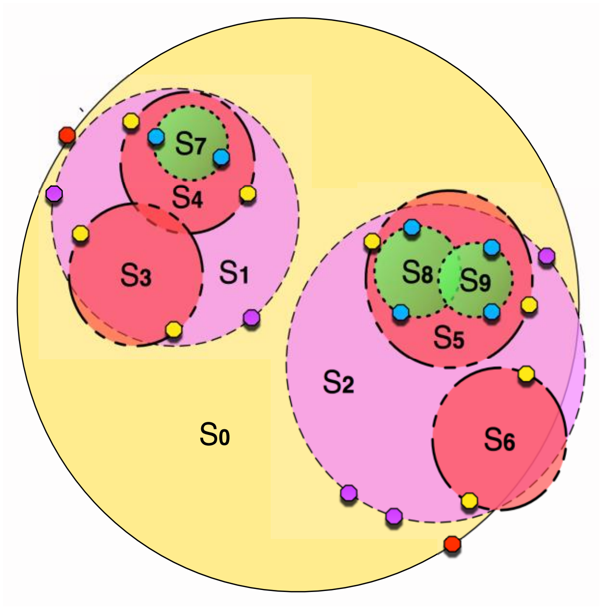


Figure 4-1 (b). kdS-tree for 21 Grids: the Circles

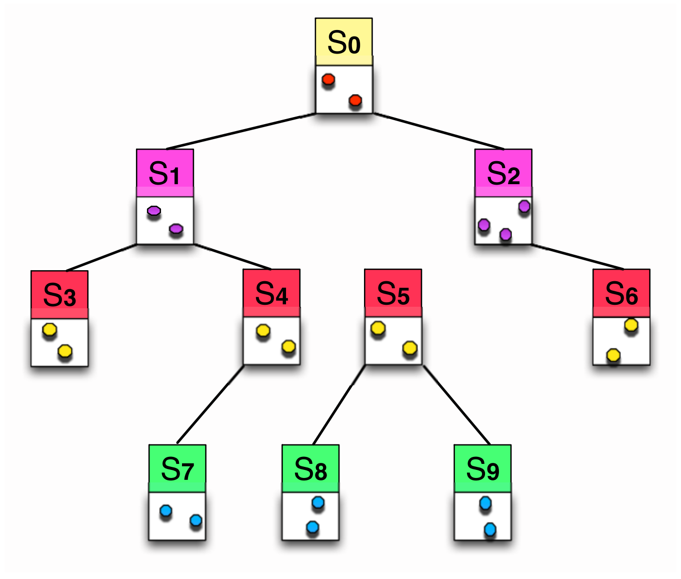


Figure 4-1 (c). kdS-tree for 21 Grids: the kdS-tree

To retrieve k nearest grids for a given taxi, the query starts with traversing the tree from root to locate the lowest level node that contains the location of the requested taxi. Meanwhile, all the distances between grids on visited nodes and location of the taxi are computed. The top-k nearest distances is then recorded. Then a query circle will be centered at the location of the taxi with a radius equal to the current k-th nearest distance. Thus, an upper bound for searching the k nearest grids can be given by the radius of query circle. The procedure will recursively back up the tree from leaf to root. If the current visited node has a grid closer than the current upper bound, the upper bound is updated. To speed up the query process, two pruning rules are proposed based on the branch-and-bound technique in [56]. The two rules are as follows:

Note that S_i represents the circle correlated to the tree node, $r(S_i)$ is the radius of S_i, Q represents the query circle of the requested taxi, $r(Q)$ is the radius of Q, g_j is a grid point located on the S_i, $dist(S_i, Q)$ is the distance between the center of S_i and the center of Q, and $dist(g_i, Q)$ is the distance between g_j and the center of Q:

Downward Pruning: If $dist(S_i, Q) > r(S_i) + r(Q)$, then for each grid g_j on S_i, $dist(g_j, Q) > r(Q)$. Therefore, the tree node of S_i can be pruned.

The Downward Pruning rule removes tree node whose circle does not intersect the query circle. If node S_i does not intersect the query circle, then there could be no nearer grids on the S_i and it is not possible for any of the subtree nodes of S_i to intersect the query circle. So the entire branch of node S_i can be pruned.

Upward Pruning: If $dist(S_i, Q) < r(S_i) - r(Q)$, then for each grid g_j on the correlated circle of the node S_i, $dist(g_j, Q) > r(Q)$. Therefore, S_i can be pruned.

The Upward Pruning rule removes tree node whose circle completely contains the query circle. If query circle is complete inside the circle of node S_i, S_i cannot have grids closer than any of current k nearest grids as well as its ancestor nodes. So node S_i and its upper tree nodes can be pruned.

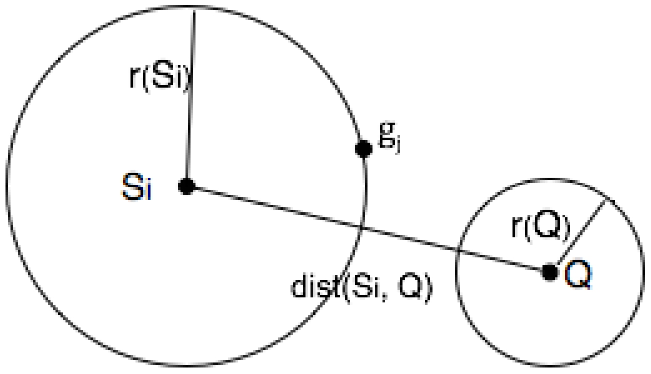


Figure 4-2 (a). Downward Pruning

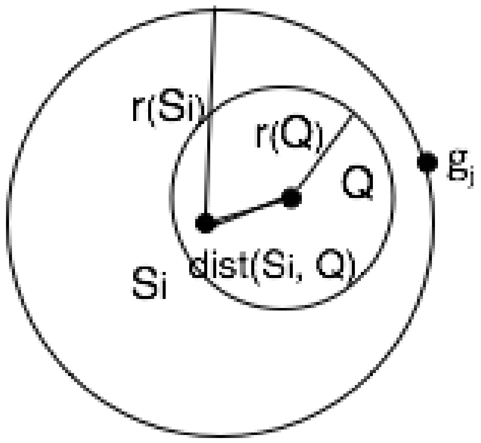


Figure 4-2 (b). Upward Pruning

Figure 4-2 shows how these pruning methods used to speed up the query. Meanwhile, a heap H of size K is used to hold the current candidates for the kNN and allows fast check and insertion.

Since the profitable potential grids changes over the time, the grid networks need to be updated.

To add a new profitable grid, the update algorithm starts from the root node and moves down the tree recursively and goes left or right child depending on whether the new grid is closer to the center of "left" or "right" circle of the splitting plane. Once the algorithm reaches to the tree node R under which the new grid is located, it checks whether the new grid is on the circle of R or inside the circle of R. If the new grid is located on the circle, then it adds the new grid to node R. Otherwise the new grid is assigned to the closer child node of R, saying CR, and a replacement node P whose minimum enclosing circle contains the new grid and the node CR is generated. Then P replaces the CR and CR becomes the child node of P.

To remove a profitable grid, the update algorithm starts from the root node and moves down the tree recursively and finds the tree node R which the grid is located on. If there is no other grid node exists in R, the algorithm removes the grid as well as the node R, and a new node P with a minimum enclosing circle is generated for child nodes of R. The node R then is replaced with P. Otherwise, the algorithm removes the grid from the R directly.

4.2 Parallel ASECR Algorithm Based on MapReduce

Due to the large volume of GPS trajectories and the complex algorithms being used to process them, traditional sequential computing models cannot handle such data with a reasonable processing time. Thus, parallel and distributed models, such as MapReduce, provide potential for processing a large set of spatial data within an acceptable response time. Therefore, in this thesis, subsection 4.2.1 briefly introduces MapReduce and its working mechanism. Then, subsection 4.2.2 describes the architecture of the proposed MapReduce Model and gives special attention to the MapReduce-based ASECR algorithm.

4.2.1 MapReduce Programming Model

MapReduce is a programming model and computing platform that is well-suited for parallel computation. In MapReduce, a program is divided into two phases: the Map phase and the Reduce phase, which are user-defined, and each phase is done in parallel on sets of <key, value> pairs. The Map phase takes in a function and a section of data as its input and applies the function to each value of the input data, which generates output data. The Map output is also a set of records that take the form of <key, value> pairs. The output data will be entered into Reduce phase. Output data with the same key will be sent to the same Reduce phase. The Reduce phase will merge these data with the same key into one set of data. The Reduce stage can only start when all the data from the Map phase is transferred into the appropriate Reduce stage. Figure 4-3 illustrates MapReduce framework.

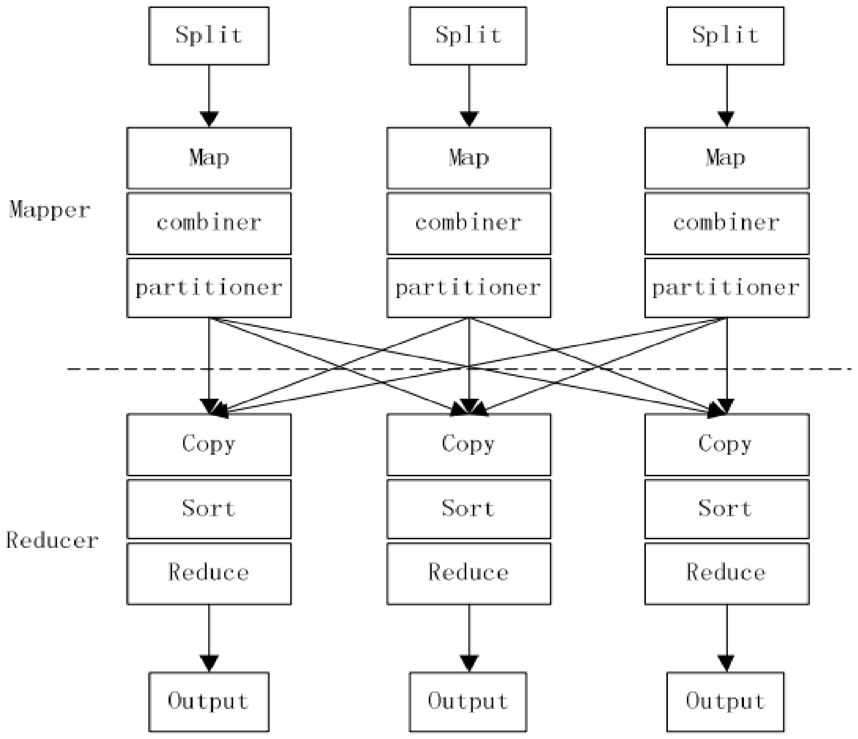
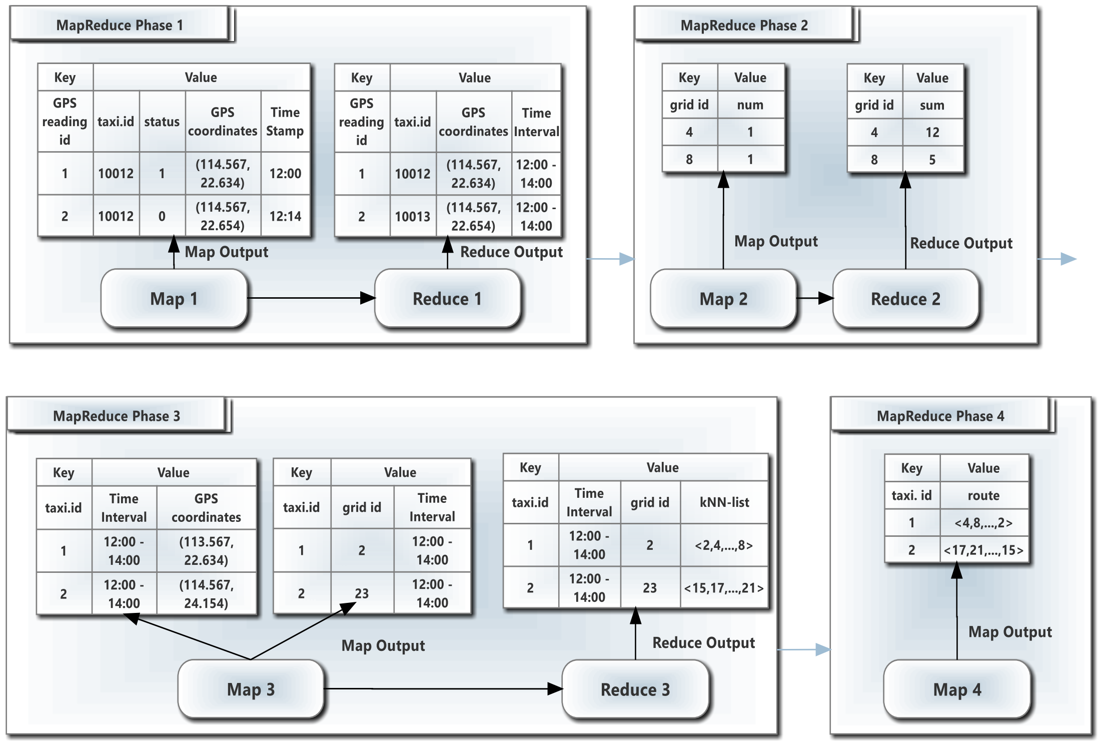


Figure 4-3. MapReduce Framework [57]

4.2.2 MapReduce for ASECR algorithm

As introduced in Chapter 3, the ASECR algorithm can be divided into four steps. The first step is data preprocessing. In this step, the pickups that fall into each grid are extracted. The second step produces kNN profitable candidate grids for each taxi. It first partitions entire space into cells, and in each cell the number of pickups in each grid is counted, then according to the information, kNN profitable grids for each taxi can be determined by using kdS-tree data structure. The third step updates kNN profitable grids. It first performs the identification of overlapping cells, which are overlapped with the query circle of the starving taxi. Then all overlapping cells are merged into one cell and a new kdS-tree is built for this new cell. After that, the kNN potential profitable grids will be updated for each starving taxis. The last step calculates shortest expected cruising route for each taxi.



Key	Value			
GPS reading id	taxi.id	status	GPS coordinates	Time Stamp
1	10012	1	(114.567, 22.634)	12:00
2	10012	0	(114.567, 22.654)	12:14

Key	Value		
GPS reading id	taxi.id	GPS coordinates	Time Interval
1	10012	(114.567, 22.634)	12:00 - 14:00
2	10013	(114.567, 22.654)	12:00 - 14:00

Key	Value
grid id	num
4	1
8	1

Key	Value
grid id	sum
4	12
8	5

Key	Value	
taxi.id	Time Interval	GPS coordinates
1	12:00 - 14:00	(113.567, 22.634)
2	12:00 - 14:00	(114.567, 24.154)

Key	Value	
taxi.id	grid id	Time Interval
1	2	12:00 - 14:00
2	23	12:00 - 14:00

Key	Value		
taxi.id	Time Interval	grid id	kNN-list
1	12:00 - 14:00	2	<2,4,...,8>
2	12:00 - 14:00	23	<15,17,...,21>

Key	Value
taxi. id	route
1	<4,8,...,2>
2	<17,21,...,15>

Figure 4-4. Workflow of MapReduce Phases

In order to efficiently process huge GPS data as well as provide real-time routes generation, each step above will be implemented as one MapReduce phase. Four

MapReduce phases are executed in sequence, as shown in Figure 4-4. In the following, each MapReduce phase will be described separately:

4.2.2.1 Pickup Extraction Phase

This MapReduce phase is a preprocessing step which responsible of calculating the sum of pick-ups for each grid, and its output data will be provided as an additional input data by subsequent MapReduce phases. Since the pickup distribution for the proposed grid network varies from different time of the day. In order to provide a better profitable route recommendation based on the current time and location information, this MapReduce phase will uniformly divide the historical GPS trajectories into a set of splits based on spatial-temporal attributes and then assign them to different computer nodes. Thus in the Map step, the trajectory dataset (as shown in Table 4-1, where key is GPS reading id and values are consisted of taxi id, taxi statue, GPS coordinates and GPS time stamp) will be divided into 12 time slots with 2-hour time interval. Then, when the status of taxi changed from 0 to 1， the pickup will be recorded for the corresponding grid. The key of Map function's output data is the $< l_i, [\tau 1, \tau 2] >$ pair, where l_i represents the corresponding grid and $[\tau 1, \tau 2]$ is the time interval that pickup happens. The value is 1, since each status change from 0 to 1 is counted as one pickup in grid l_i during the time interval $[\tau 1, \tau 2]$. The example output format of Map function is shown in Table 4-2. The Reduce function receives the key-value pairs from the Map step. The pickups in the same grid and time slot are sent to the same Reduce job. It merges key-value pairs with the same key and sums up the number of pickups for each grid. The format of the output data of Reduce step is shown in Table 4-3.

Table 4-1. Input Data Format

Key	Value			
GPS reading id	Taxi.id	Status	GPS coordinates	Time Stamp
1	10012	1	(114.567, 22.634)	12:00
2	10012	0	(114.256, 22.659)	12:14

Table 4-2. Output Data Format of Map Job in Phase 1

Key	Value
<Grid id, (Time Interval)>	Pick-up Time
<4, (12:00 - 14:00)>	1
<4, (12:00 - 14:00)>	1
<4, (14:00 - 16:00)>	1
<8, (12:00 - 14:00)>	1
<8, (12:00 - 14:00)>	1

Table 4-3. Output Data Format of Reduce Job in Phase 1

Key	Value
<Grid id, (Time Interval)>	Sum
<4, (12:00 - 14:00)>	12
<8, (12:00 - 14:00)>	5

<u>4.2.2.2 kNN Potential Profitable Grid Candidate Generation Phase</u>

This phase decompose the entire space into cells based on the spatial distribution of trajectories and each cell will be assigned to one computer node for processing. Since the potential profitable grids are unequally distributed on the grid network, simply decomposing the entire space into equal–sized cells and sending equal–sized cells to computer nodes will break computer nodes' load balance and let some computer nodes end up not getting enough assigned potential profitable grids. In order to avoid unbalanced load problem, the large GPS data is separated into smaller GPS data sets offline based on the spatial distribution of the profitable grids and make each node has more than k potential profitable grids. Specifically, the whole space is first divided into four equal–sized cells, then each cell will be checked on the number of potential profitable grids contained. Cells with more than k profitable grids will be divided into four smaller equal–sized cells. The process continues until each cell contains less than k profitable grid. If a cell is with less than k profitable grids, it will be merged with its three neighbour cells with the same size, and then the space partition will stop. Figure 4-5 shows an example to show the partition output of GPS trajectories when k equals to 3, where the red circle presents the potential profitable grids. As we can see in the figure, the grid network is divided into different cell size according to the density of potential profitable grids, the cell which contains less than 3 grids will be merged into bigger cell with its neighbours. Then the entire space is decomposed into unequal–sized cells $\{C_1, C_2, \dots, C_n\}$ based on grids' distribution, which are then assigned to the corresponding computer nodes.

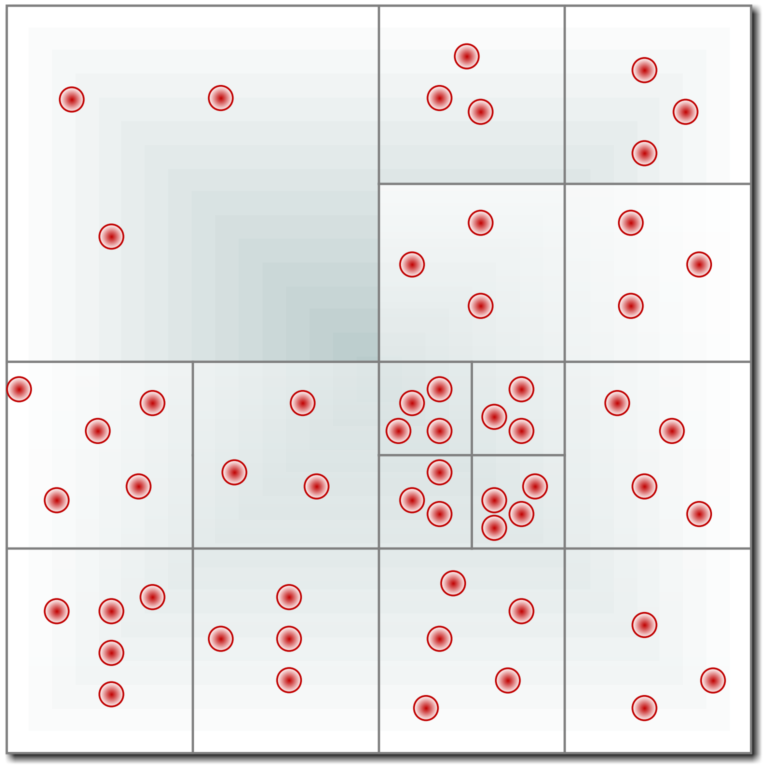
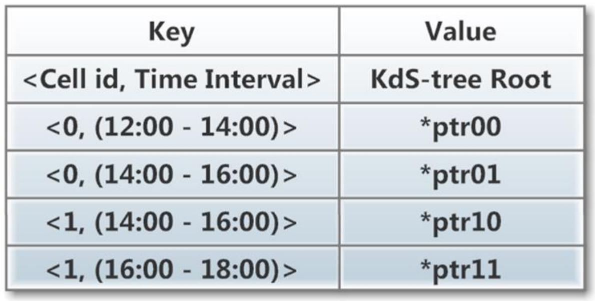


Figure 4-5. An Example of GPS Trajectories Partition

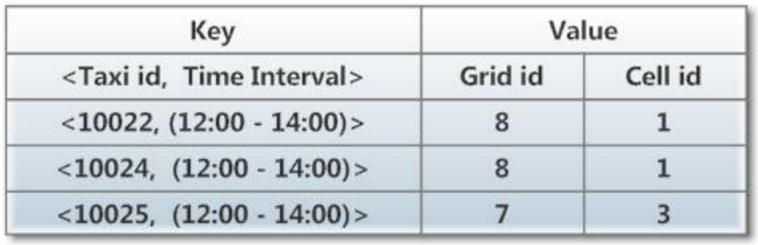
Since two datasets are utilized in this phase, one is the historical GPS trajectory dataset for grid network operations and the other is the real time GPS-readings for each requested taxi, two Map functions are applied. The first Map function is for the grid network, which creates a kdS-tree for each cell. The output key-value pair of the function is that the key is the corresponding cell C_i and its time interval, and the value is the root node of kdS-tree. Table 4-4 shows an example of this Map function's output format: the *ptr00 is a kdS – tree root for cell 0 during the time interval (12:00 – 14:00) while the *ptr01 is another kdS – tree root for cell 0 during the time interval (14:00 – 16:00). The second Map function is for each taxi t_{id}. The key is the $< t_{id}, [\tau 1, \tau 2] >$, which indicates the taxi t_{id} locating in a grid l_i at time interval $[\tau 1, \tau 2]$, and the value consists of the grid id l_i and the cell id C_i that grid l_i located in. Table 4-5 is the output format for the second Map function. For example, the first item recorded in Table 4-5 means taxi 10022 is looking for passenger at grid 8 between 12:00 and 14:00. The Reduce function receives the records from two Map functions and estimates the kNN potential profitable grids for each taxi by querying the kdS-tree, and forms a list of possible kNN profitable grid candidates with the format $< t_{id}, \ , (l_i, \ldots, l_j\ ) >$, where l_i the profitable grid is assigned to taxi t_{id} at time interval $[\tau 1, \tau 2]$. Finally, Reduce function generates a key-value pair in which the key is t_{id} and time interval $[\tau 1, \tau 2]$, and the value is kNN candidates list and the cell id C_i which the kNN candidate grids belong to. The output format of Reduce function is shown in Table 4-6. For each requested taxi, the recommended route will be generated with respect of current time interval. Figure 4-6 shows the workflow of this phase.

Table 4-1. Output Data Format of Map Job 1 in Phase 2



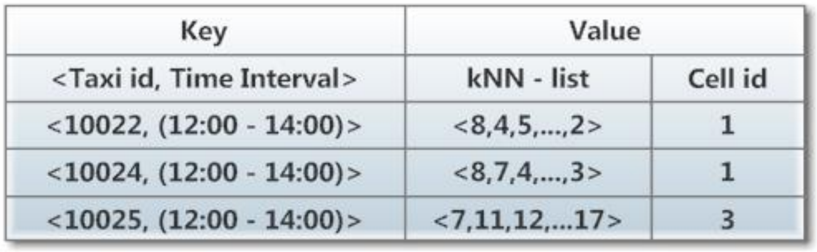
Key	Value
<Cell id, Time Interval>	KdS-tree Root
<0, (12:00 - 14:00)>	*ptr00
<0, (14:00 - 16:00)>	*ptr01
<1, (14:00 - 16:00)>	*ptr10
<1, (16:00 - 18:00)>	*ptr11

Table 4-2. Output Data Format of Map Job 2 in Phase 2



Key	Value	
<Taxi id, Time Interval>	Grid id	Cell id
<10022, (12:00 - 14:00)>	8	1
<10024, (12:00 - 14:00)>	8	1
<10025, (12:00 - 14:00)>	7	3

Table 4-3. Output Data Format of Reduce Job in Phase 2



Key	Value	
<Taxi id, Time Interval>	kNN - list	Cell id
<10022, (12:00 - 14:00)>	<8,4,5,...,2>	1
<10024, (12:00 - 14:00)>	<8,7,4,...,3>	1
<10025, (12:00 - 14:00)>	<7,11,12,...17>	3

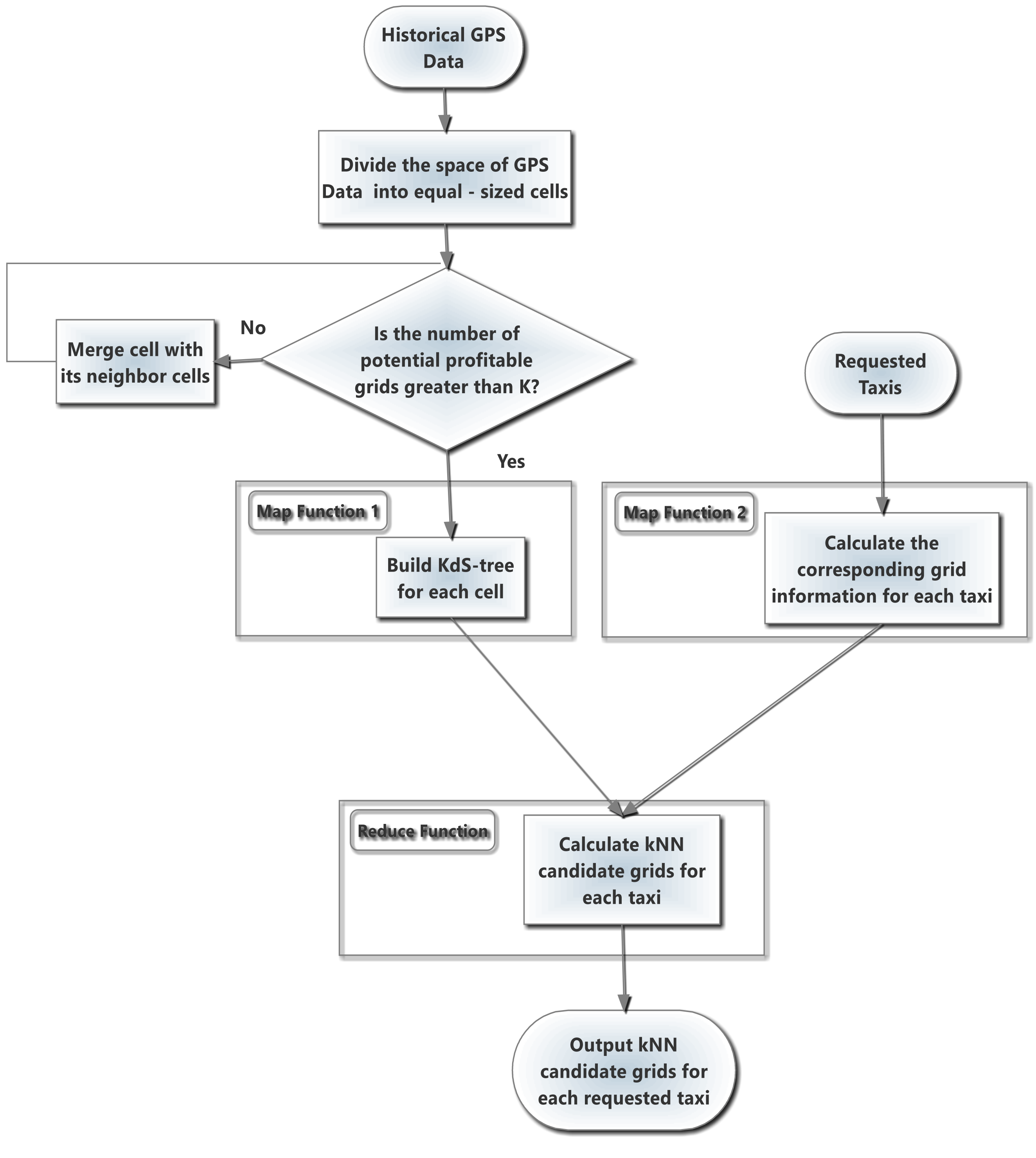

Historical GPS Data
Divide the space of GPS Data into equal - sized cells
Is the number of potential profitable grids greater than K?
No
Merge cell with its neighbor cells
Yes
Requested Taxis
Map Function 1
Build KdS-tree for each cell
Map Function 2
Calculate the corresponding grid information for each taxi
Reduce Function
Calculate kNN candidate grids for each taxi
Output kNN candidate grids for each requested taxi

Figure 4-6. Flowchart for Generating kNN Profitable Candidate Grid

4.2.2.3 kNN Profitable Grids Update Phase

The purpose of the kNN profitable grids update phase is to update the top k nearest distances for each taxi. Since in the second MapReduce phase, the k potential profitable grids are only generated from the cell that taxi located at, there is a chance the nearby cells may contain closer potential profitable grids than the current top k nearest profitable grids. So in this phase, a query circle is generated, which uses the location of the taxi as the center and the distance to the current k-th nearest profitable grids as the radius. Then the overlaps between neighboring cells and the query circle are checked. The kNN profitable grids will be updated if overlaps is detected. Figure 4-7 shows the flowchart of the phase. As shown in the figure, the Map function identifies the overlapping cells with the query circle for each taxi. If no overlap occurs, which means there is no top k nearest profitable grids in the other cells. Then there is no need to perform any additional step to update the kNN candidate grids list and the output record of Reduce function from the previous phase will be directly sent to the next MapReduce phase. Otherwise, all cells overlapping with the query circle are merged into a region and a kdS-tree is built for the region. The output key-value pair of this Map function is that the key is the corresponding region R_i and its time interval, and the value is the new root node of the bigger kdS-tree. Table 4-7 shows the output data format of this Map function. Then the Reduce function receives two records: one is the output information of Map function from the present phase and the other is the output information of the second Map function from previous phase, which records the grids id as well as the cells id that requested taxis located in at time interval $[\tau 1, \tau 2]$. Next, the Reduce function will repeat the operation as the previous Reduce phase and generate a key-value pair in which the key is taxi id t_{id} and time interval $[\tau 1, \tau 2]$, and value is the updated list of the k potential profitable grids and the new cell id C_i.

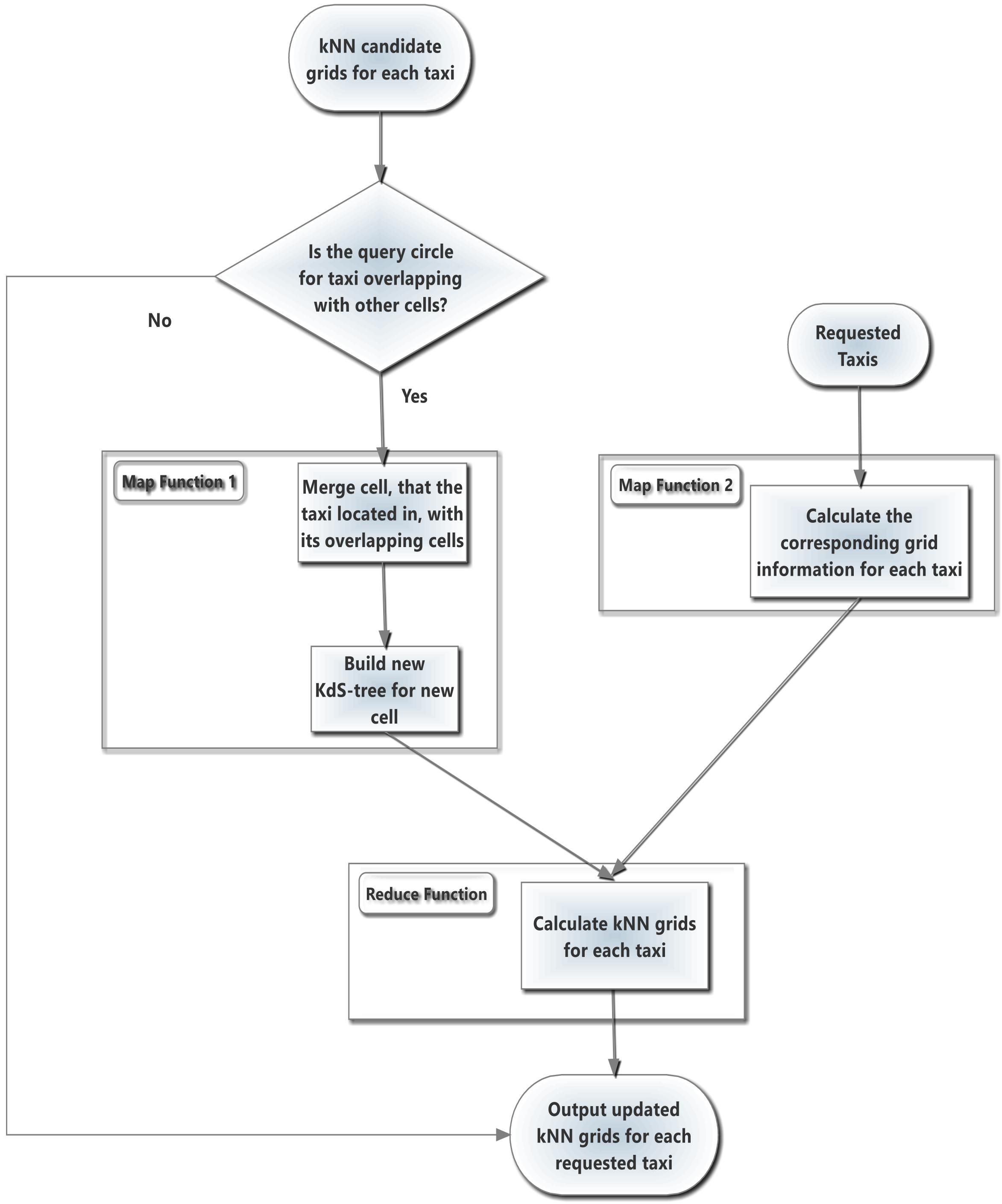
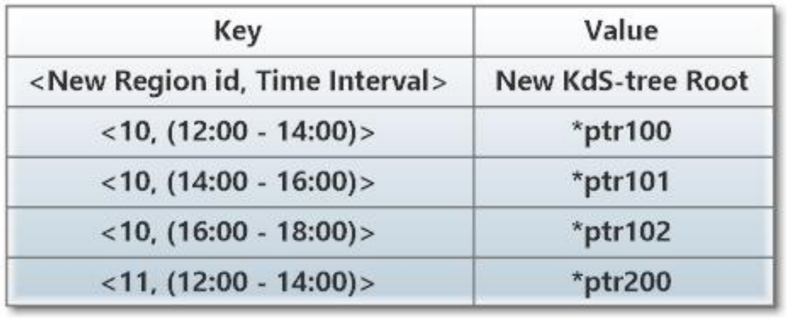

kNN candidate grids for each taxi
Is the query circle for taxi overlapping with other cells?
No
Yes
Requested Taxis
Map Function 1
Merge cell, that the taxi located in, with its overlapping cells
Build new KdS-tree for new cell
Map Function 2
Calculate the corresponding grid information for each taxi
Reduce Function
Calculate kNN grids for each taxi
Output updated kNN grids for each requested taxi

Figure 4-7. Flowchart for Updating kNN Profitable Grids

Table 4-7. Output Data Format of Map Job in Phase 3

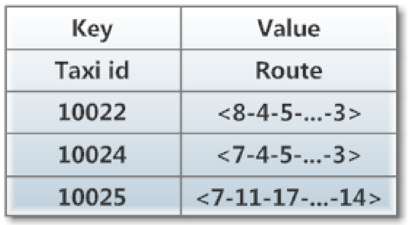


Key	Value
<New Region id, Time Interval>	New KdS-tree Root
<10, (12:00 - 14:00)>	*ptr100
<10, (14:00 - 16:00)>	*ptr101
<10, (16:00 - 18:00)>	*ptr102
<11, (12:00 - 14:00)>	*ptr200

4.2.2.4 Routing Recommendation Phase

The last phase is a Map-only phase since the shortest expected cruising route will be calculated for each individual taxi and there is no need to consolidate or aggregate individual results. The Map function receives the kNN profitable grid candidates of each taxi and output a potential profitable route by calculating the expected cruising distance of all the possible sequences in the K nearest profitable assigned grids. The one with the shortest expected distance will be selected as the recommended profitable route based on the K assigned grids. The output data format is shown in Table 4-8, where the key is taxi id and the value is the recommended route, which consists of a sequence of potential profitable grids, provided to each taxi.

Table 4-8. Output Data Format of Reduce Job in Phase 4



Key	Value
Taxi id	Route
10022	<8-4-5-...-3>
10024	<7-4-5-...-3>
10025	<7-11-17-...-14>

CHAPTER FIVE: EXPERIMENTS

This section evaluates the two proposed algorithms, SECR and ASECR, on two real taxi datasets. Map-Reduce is implemented based on Hadoop version 2.5.2 and Java 1.6.0.29. All experiments are conducted running on PCs with a 2.0 GHz dual processor and 2 GB of memory. All of the reported results have an average of 20 runs.

5.1 Description of Datasets

Taxi GPS trajectories are collected from taxis with GPS equipment and report their locations (Longitude, Latitude) with certain intervals, including their taxi status (with or without passengers). Two real taxi GPS trajectory datasets are used for the experiments. One set is from San Francisco, USA and the other set is from Shenzhen, China. The San Francisco dataset contains GPS trajectories of more than 500 taxis in a 25-day period, and the Shenzhen dataset contains GPS trajectories of more than 13,000 taxis in a 30-day period. The Shenzhen dataset is over 73 GB, and the San Francisco dataset is over 170 MB. Each record consists of following information:

TAXI_ID: the unique ID of each taxi;

DATA: the sample timestamp “MM, DD, YYYY”

TIME: the sample timestamp “HH: MM:SS”;

LONGITUDE: the direction the taxi is heading in, from an angular measurement ranging from 0° at the Prime Meridian to +180° eastward and −180° westward;

LATITUDE: the direction the taxi is heading in, from 0° at the Equator to 90° (North or South) at the poles;

TAXI_STATE: indicates whether the taxi has a passenger or not: 1 if the taxi is occupied and 0 if it is empty.

An example of the GPS trajectory table is showed as below.

Table 5-1. A Sample of GPS – Reading

Field	Instance	Memo
TAXI_ID	D2463	Unique identification of taxi
DATA	20120115	Jan. 15th, 2012
TIME	150230	15:02:30
LONGITUDE	-114.114117	Degree
LATITUDE	22.550917	Degree
TAXI_STATE	1	0 represents taxi with no passengers, 1 represents taxi with passengers.

The grid networks are generated using 2-hour intervals for both datasets. The grid size is set to 550 meters for San Francisco dataset and 350 meters for Shenzhen dataset by using the grid determination method discussed in Section 3.2.

5.2 Parameter Study on the Number of Potential Profitable Grids K

K is the number of potential profitable grids assigned to each taxi to generate the profitable route. Since the passenger's location cannot be predicted, it is difficult to predict how many potential profitable grids should be assigned to each taxi. In this experiment, the distance performance of taxis changing over the different values of K is investigated. The parameter K is set from 2 to 12. Figure 5-1 shows a progressive increase of the distance performance until some certain K value, beyond which the distance performance levels off. The reason is when K increases, more assigned profitable potential grids lead to higher pick-up probability. Therefore, the distance performance for taxis increases with decreasing cruising distances. However, after the certain value (e.g. when more than 8 grids for San Francisco dataset) is assigned to each taxi, the distance performance stabilized. The reason is that most of the taxis would have already picked up passengers after visiting K number of potential profitable grids. From the figure, we can tell that the level off value for San Francisco dataset is 8 grids, which indicates assigning more than 8 potential grids does not improve taxis distance performance very much. Similar observations can be found from Shenzhen dataset, which the level off K value is 6.

The running time of querying the K potential profitable grids with respect to different K values is also investigated. As shown in Figure 5-2, the running time increases with the increasing of K value. In practice, the distance performance and the computation time needs to be balanced. So, this experiment set K to 8 for San Francisco dataset and 6 for Shenzhen dataset for the rest of experiments.

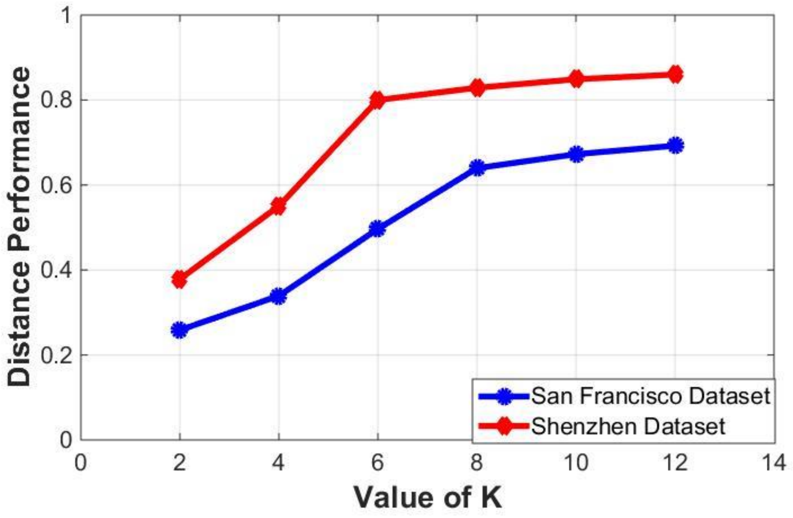


Figure 5-1. Distance Performance vs. K

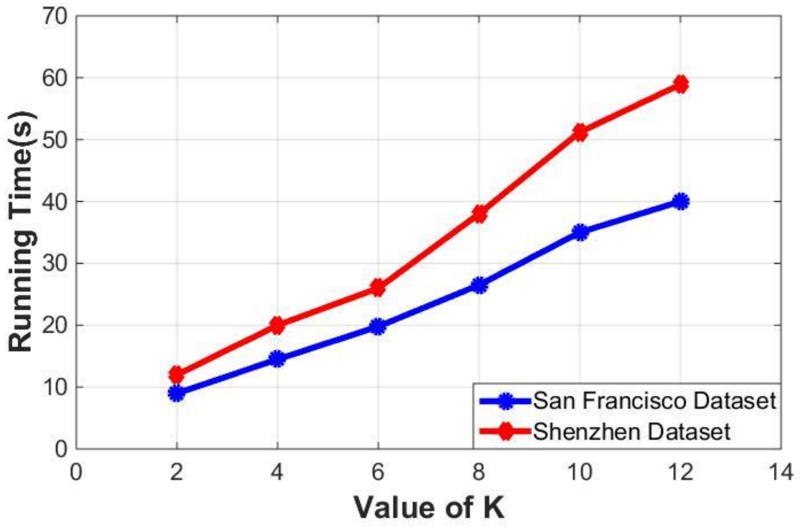


Figure 5-2. Running time vs. K

5.3 KdS-tree performance Pruning Effectiveness and Query Efficiency

This experiment first compares the query efficiency of the proposed kdS-tree over the kd-tree. The query efficiency is defined as the difference of the query response CPU time between kdS-tree and kd-tree over the query response CPU time of kd-tree:

Definition 13: Given a grid l where taxi is located, a root nodes $r_{kd-tree}$ for kd-tree and a root nodes $r_{kdS-tree}$ for kdS-tree in time interval $[\tau 1, \tau 2]$, the ***query efficiency*** is defined as:

$$e_l = \frac{|\tau(r_{kd-tree}, l) - \tau(r_{kdS-tree}, l)|}{\tau(r_{kd-tree}, l)},$$

where e_l is the ***query efficiency*** for taxi's location l , $\tau(r_{kd-tree}, l)$ is the CPU query response time of kd-tree and , $\tau(r_{kdS-tree}, l)$ is the CPU query response time of kdS-tree.

Figure 5-3 shows the query efficiency over different K values. The figure shows that kdS-tree outperforms the kd-tree and the query efficiency increases as K increases. The reason is that the radius of query circle extends as K increases, and the larger query circle leads to more nodes need to be examined by kd-tree than kdS-tree due to the corner of kd-tree's rectangle boundary will make it easier to intersect with a larger query circle.

As mentioned, two pruning rules are proposed to speed up the query process by reducing the number of tree nodes which needed to be checked. The effectiveness of the proposed pruning rules is also evaluated on the kdS-tree. In this experiment, two kdS-trees are built for both datasets: the kdS-tree of Shenzhen dataset contains 6612 nodes and the one of San Francisco dataset contains 1862 nodes. A visited flag is used for each node to record if the node has been visited. During the querying process, the number of visited tree nodes are summed up and recorded for both using pruning rules method and without using pruning rules method. Table 5-2 shows the average number of visited kdS-tree nodes for one taxi to find K nearest profitable grids with and without the two pruning rules. The number of potential profitable grids K varies from 2 to 12. As expected, the pruning rules can reduce the number of visited nodes. For example, when K is set to 8, 197 nodes are visited with pruning rules while 361 nodes are visited without pruning rules. It shows that with two pruning rules, about 45% of the nodes can be eliminated during the search process.

Table 5-2. The Number of kdS Tree Nodes Visited With and Without Pruning Rules

Value of K	San Francisco Dataset		Shenzhen Dataset	
	Before Pruning	After Pruning	Before Pruning	After Pruning
2	13	11	16	14
4	52	31	89	67
6	108	53	562	388
8	361	197	1498	864
10	776	397	3168	2082
12	1298	695	4398	3602

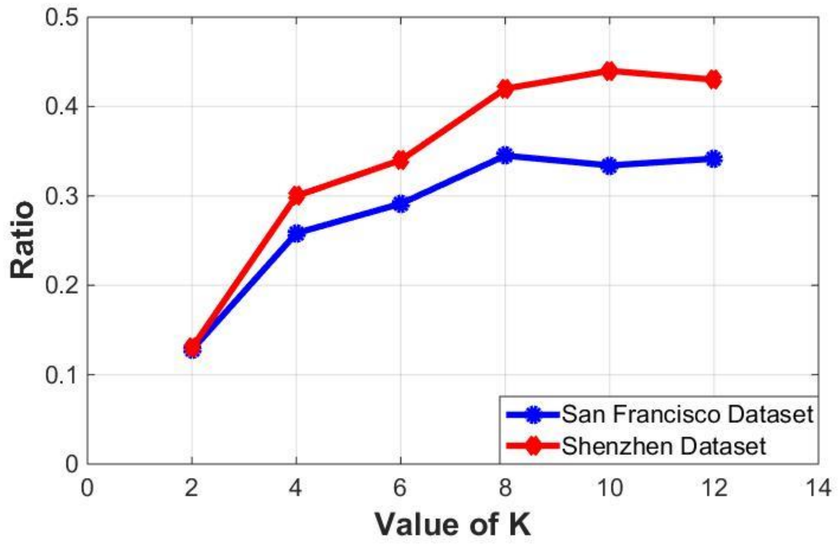


Figure 5-3. Query Efficiency vs. K

5.4 Evaluation on Recommendation

This experiment compares the time and distance performance of the ASECR algorithm and the SECR algorithm with two other methods: the LCP [1] method and the baseline method. Here, the length of route for LCP method is set to the same value as the number of potential profitable grids for SECR and ASECR. The baseline method recommends all taxis to their nearest profitable grid with respect of each grid's capacity. The experiment evaluates algorithms by simulating the real world events on every 2-hour time window of a day. In addition, the time performance of taxis is also investigated, which is the time a taxi travels while it is occupied divided by the whole time that taxi is cruising. A larger value of time performance reflects better time efficiency of the taxi.

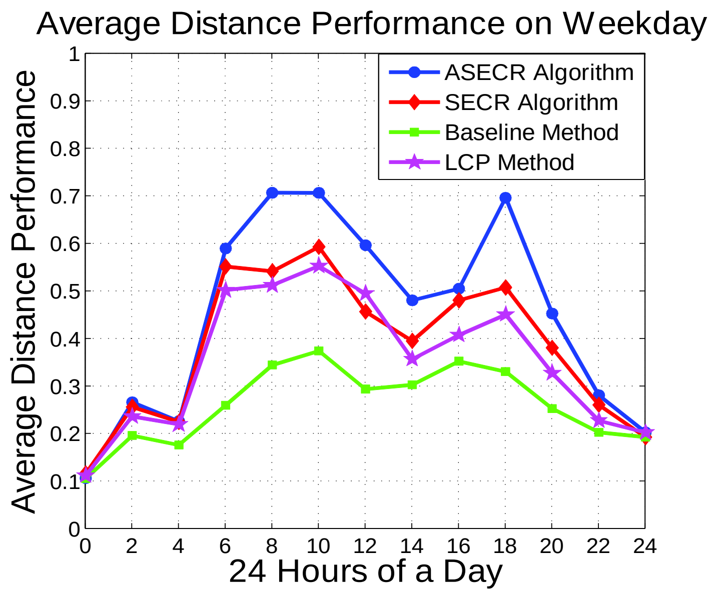


Figure 5-4. San Francisco: Average Distance Performance of Weekday

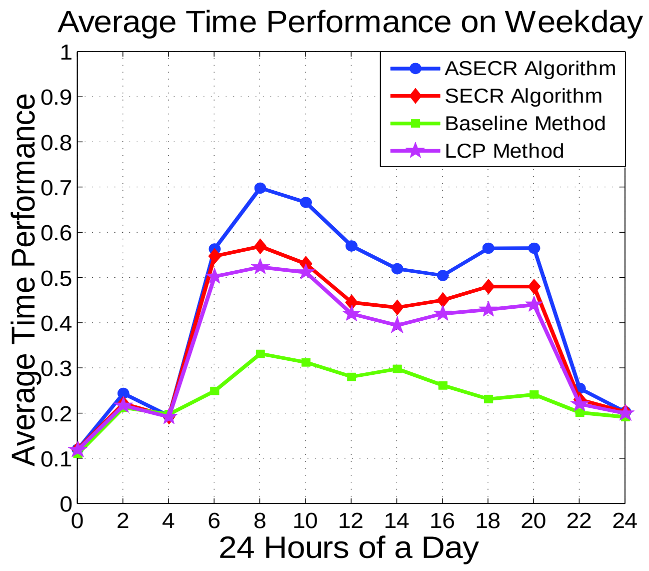


Figure 5-5. San Francisco: Average Time Performance of Weekday

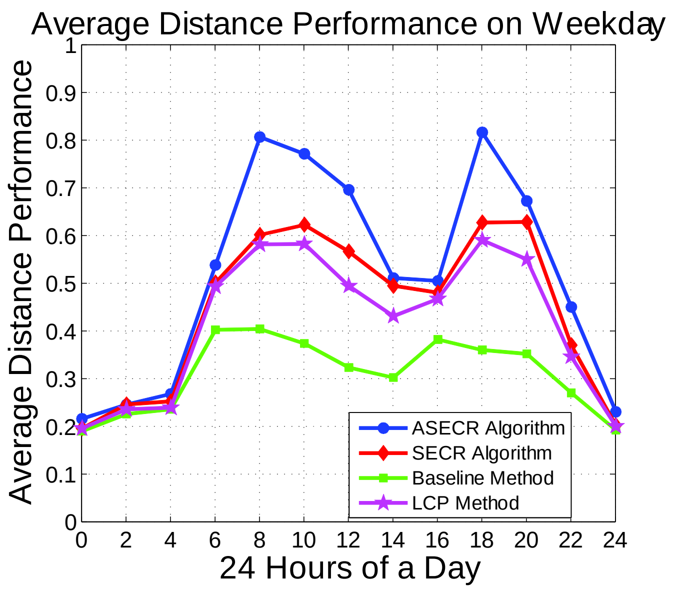


Figure 5-6. Shenzhen: Average Distance Performance of Weekday

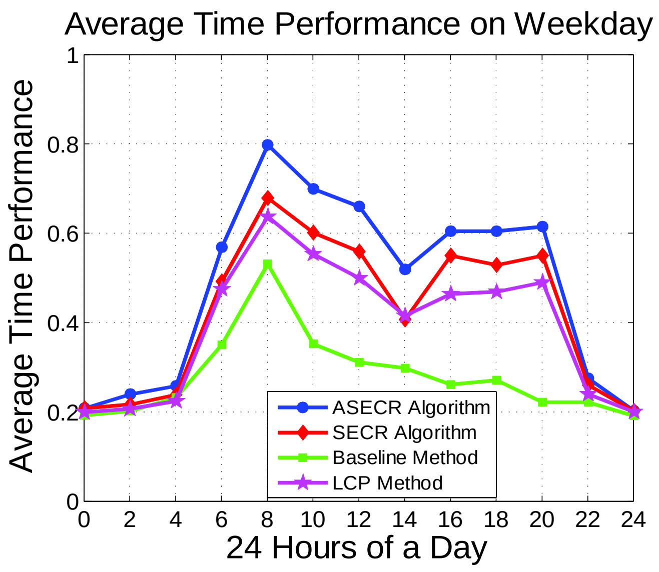


Figure 5-7. Shenzhen: Average Time Performance of Weekday

The overall time and distance performance of the algorithms on weekday for San Francisco dataset are shown in Figures 5-4 and 5-5. From the figures, The ASECR algorithm outperforms SECR algorithm, LCP method and baseline method in term of distance and time performance of taxis. One important observation from Figures 5-4 and

5-5 is that both SECR and ASECR outperform baseline and LCP methods during both rush hours (i.e. 8am to 10am, and 4pm to 6pm) and non-rush hours, which show the effectiveness of two algorithms. This is because during rush hours, the SECR and ASECR algorithms are designed to assign grids with more passengers to more taxis, which leads to the significant improvement for both distance performance and time performance by avoiding the passengers' competition. In addition, ASECR outperforms SECR by about 25% (on average) on average during rush hours. This demonstrates that continually updating taxis' request locations and times can help improve the profitable route recommendation. The other observation from Figures 5-4 and 5-5 is that during the rush hours of a weekday, the percentage of distance performance and time performance are both higher than that of non-rush hours. This is because during rush hours, there are more passengers, which indicates more pick-up opportunities during rush hours. The same trend of overall time and distance performance for San Francisco dataset are shown in Figures 5-6 and 5-7.

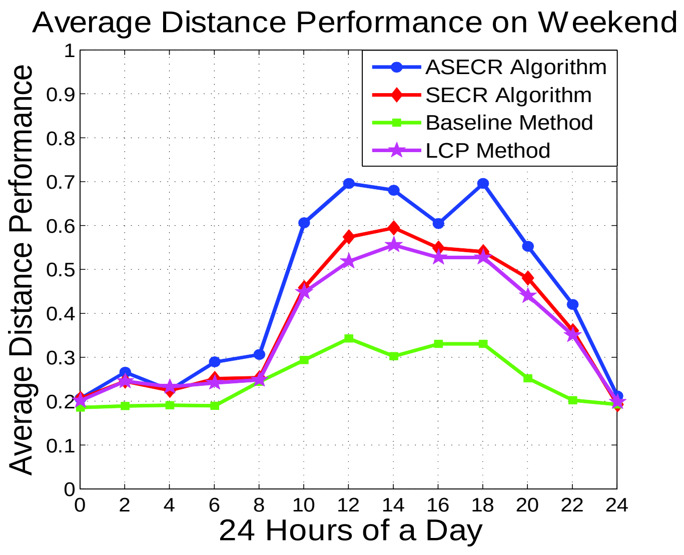


Figure 5-8. San Francisco: Average Distance Performance of Weekend

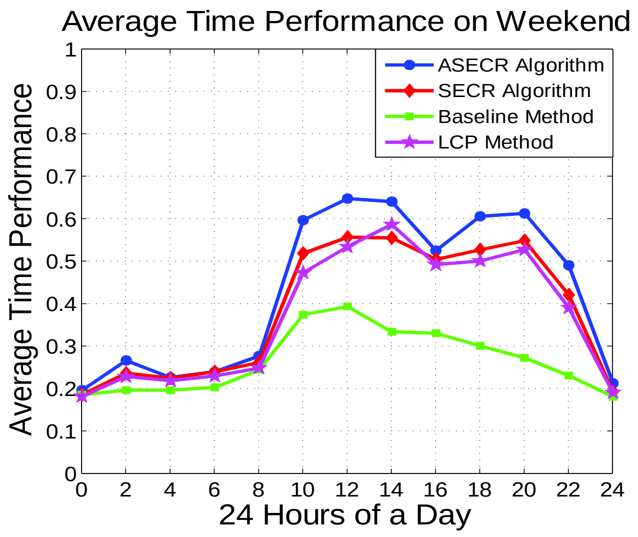


Figure 5-9. San Francisco: Average Time Performance of Weekend

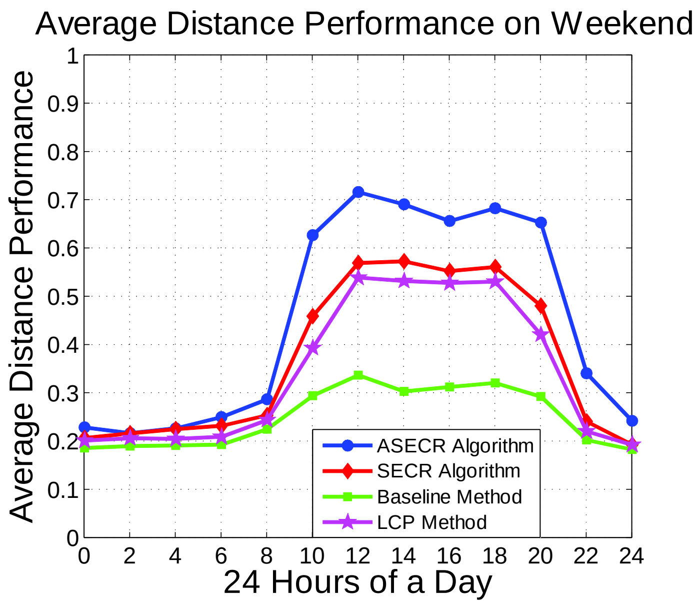


Figure 5-10. Shenzhen: Average Distance Performance of Weekend

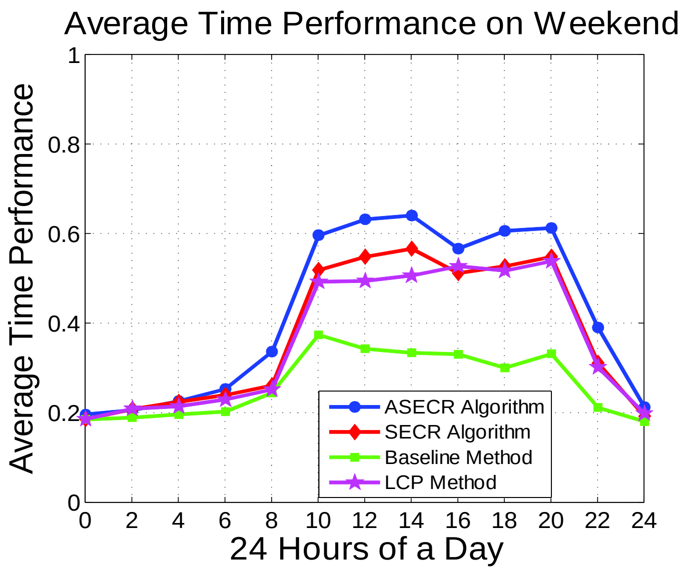


Figure 5-11. Shenzhen: Average Time Performance of Weekend

Since people take taxis at different time on weekends than weekdays, the recommended route should be different from weekdays even at the same time interval. A further investment is performed on weekend dataset to provide better recommendation routes for taxis. Figures. 5-8, 5-9, 5-10 and 5-11 give the average distance and time performance on weekend in 2-hour time windows. Similar conclusions can be drawn from the two figures. Different than weekdays, the distance performance and time performance of ASECR and SECR are better than LCP and baseline algorithms from 10:00 to 22:00 for both San Francisco and Shenzhen data, which reflects the rush hours of weekends is different of that in weekdays.

CHAPTER SIX: CONCLUSIONS AND FUTURE WORK

In this final chapter, the conclusions are made and the further researches are discussed.

6.1 Conclusions

This thesis studies the problem of how to improve taxis' performance by minimizing the expected cruising distance for a group of taxis. It utilizes historical taxi GPS trajectories to recommend a personalized profitable route for each taxi. The proposed algorithms aim to minimize the cruising distance based on the current time and the current taxi location. Compared with other route recommendation methods in literature, our approach has the following distinctive characteristics.

Firstly, a temporal profitable grid network is built from the taxis' historical GPS data. It partitions the map into equal-sized grids, where each grid is a rectangle. Each grid is bounded by the probability of finding a passenger inside it. This grid network not only offers simplicity in implementation, but also easily shows hotspots areas of the map.

Secondly, an expected cruising distance is defined to formulate the potential cruising distance. Since there is no guarantee that the taxi can find its passenger at the first recommended grid, this thesis assigns k potential profitable grids to taxi in order to maximize the probability of finding a pick-up by going through the route that consists of the assigned grids. The expected cruising distance is calculated by integrating the distribution of the conditional probability with the distribution of the cruising distance before picking up a passenger. The most profitable route is the shortest expected cruising distance with the minimal expected cruising distance of all possible routes and this route will be recommended to the taxi.

Thirdly, In order to find the shortest expected cruising distance for taxis, two profitable routing approaches are proposed in the thesis: the SECR and ASECR algorithms. The SECR algorithm recommends a profitable route by searching the shortest expected cruising distance among the assigned profitable grids to the taxis. This algorithm provides a recommended route with potential profitable grids to each taxi , where each taxi is more likely to pick up passengers within short distance (during the routes) and may maximize the profit of the next trip. As the time and location of a taxi changes over time, the SECR algorithm is refined by proposing the ASECR algorithm, which is an adaptive route searching method based on the continuous movement of taxis.

Lastly, to improve the efficiency of the algorithm, a new data structure called kdS-tree and Map-Reduce models are implemented in the algorithm. kdS-Tree, a binary tree in which each node is associated with a circle, is extended from kd-Tree and ball-Tree. It is designed for frequently updating tree structures and efficiently queries profitable potential. In order to solve the problem of not being able to load data into memory and to avoid frequent disk access, four MapReduce phases are implemented. The first MapReduce phase is a data preprocessing phase that used to extract the pickups for each grid. The second and third MapReduce phases are designed not only to generate kNN profitable grids for taxi but also to constantly update kNN profitable grids efficiently. Then the recommended routes are produced in the last phase and will be delivered to each taxi.

The experiments on the two real taxi trajectory datasets from San Francisco and Shenzhen show the effectiveness and efficiency of the two proposed algorithms, which may provide more profitable routes to a group of taxis.

6.2 Future Work

Several extensions of this research may be regarded as priorities for future research and are listed as follows.

The taxi GPS trajectories are large spatial and temporal data. Future research should focus on evaluating the proposed algorithms on distributed large datasets. Since GPS data are often collected by different GPSs and stored accordingly on different devices, in order to deal with the distributed GPS data, the communication and synchronization between the different devices must be minimized, and an algorithm for preprocessing and integrating the original GPS data is necessary.

Since GPS trajectories also contain information like the driving speed and the speed factor of taxis, the information can be used to describe traffic conditions. Utilizing speed patterns during recommendations may be another possible research topic, which may help taxi drivers find passengers quickly and avoid traffic.

Moreover, visualization techniques can be integrated into our methods so that taxis can adjust the provided recommended routes based on their preferences, rather than just passively observing them. In this thesis, one recommended route will be automatically generated and suggested to taxis based on the proposed recommendation algorithm. However, in the real-world, taxis may have multiple travelling routes to reach the same destination. They prefer to choose routes based on the traffic, road conditions, or drivers' preferences. Furthermore, traffic conditions may change over time. To avoid travelling through busy areas and saving driving time, experienced taxi drivers can decide which assigned potential profitable grids to be used for generating the recommended route. Therefore, showing a visualized grid network with road information to taxis could help them incorporate their experiences and preferences into the analytical process, which may further improve the route recommendation efficiency, as demonstrated in the visualization-assisted route recommendation system. Hence, exploring visualization techniques may help taxis have a comparison of trends and historical patterns along with vivid GPS data. Consequently, this may bring more flexibility for taxis when choosing their travelling routes.

BIBLIOGRAPHY

[1] Y. Ge, H. Xiong, T. Alexander, K. Xiao, G. Marco and P. Michael, "An Energy-Efficient Mobile Recommender System," *The 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* Washington, DC, USA, 2010.

[2] M. R. Bloomberg and D. Yassky, Schaller Consulting, " The New York City Taxicab Fact Book, " *Schaller Consulting, Brooklyn,* NY , 2006.

[3] T. Hunter, R. Herring, P. Abbeel and A. Bayen, "Path and Travel Time Inference from GPS Probe Vehicle Data," *Neural Information Processing Systems Foundation (NIPS),* Vancouver, Canada, 2009.

[4] R. F. Meyer, and H. B. Wolfe, "The Organization and Operation of a Taxi Fleet," *Journal of Naval Research Logistics Quarterly*, pp. 137–150, 1961.

[5] K. Yamamoto, K. Uesugi and Y. Watanabe, "Adaptive Routing of Cruising Taxis by Mutual Exchange of Pathways," *International Journal of Knowledge Engineering and Soft Data Paradigms,* vol 2, pp. 57 – 69, 2010.

[6] J. Powell, Y. Huang, Bastani and J. M. F., "Towards Reducing Taxi Time Using Spatio-Temporal Profitability Maps," *The 12th International Conference on Advances in Spatial and Temporal Databases,* Minneapolis, pp. 242 – 260, 2011.

[7] Sanders and Schultes, "Highway hierarchies hasten exact shortest path queries," *The 13th Annual European Conference on Algorithms,* Spain, pp. 568 – 579, 2005.

[8] R. Gotsman, "Generating Map-based Routes from GPS Trajectories and their

Compact Representation," *Technion - Israel Institute of Technology, Taub*, 2012.

[9] J. Yuan, Y. Zheng, L. Zhang, X. Xie and G. Sun, " Where to Find My Next Passenger?," *IEEE Transactions on Knowledge and Data Engineering,* Beijing, China, 2011.

[10] H.W. Chang, Y.C. Tai, H.W. Chen and J.Y.J. Hsu, "iTaxi: Context-Aware Taxi Demand Hotspots Prediction Using Ontology and Data Mining Approaches," *International Journal of Business Intelligence and Data Mining,* Chung-Li, Taiwan, 2008.

[11] O. Wolfson, B. Xu, S. Chamberlain and L. Jiang, "Moving Objects Databases: Issues and Solutions," *International Conference on Scientific and Statistical Database Management*, Washington, DC, USA, pp. 111 – 122, 1998.

[12] H. Güting, . T. d. Almeida and Z. Ding, "Modeling and querying moving objects in networks," *The International Journal of Very Large Data Bases* , vol 15, pp. 165-190, 2006.

[13] S. Brakatsoulas, . D. Pfoser and N. Tryfona, "Modeling, Storing, and Mining Moving Object Databases," *Journal of Database Engineering and Applications Symposium*, pp. 68 – 77, 2004.

[14] A. Kesting and M. Treibe, "Calibrating Car-Following Models by Using Trajectory Data: Methodological Study," *Transportation Research Record: Journal of the Transportation Research Board,* vol. 2088, pp. 148-156, 2008.

[15] J.G. Lee, J. Han, X. Li and H. Gonzalez, "TRACLASS: Trajectory Classification

Using Hierarchical Region-Based and Trajectory-Based Clustering," *International Conference on Very Large Data Base*, Auckland, New Zealand, 2008.

[16] C. Sung, D. Feldman and D. Rus, "Trajectory Clustering for Motion Prediction," *International Conference on Intelligent Robots and Systems (IROS)*, Portugal, pp. 1547 - 1552, 2012.

[17] M. Nanni and D. Pedreschi, "Time-focused clustering of trajectories of moving objects," *Journal of Intelligent Information Systems*, vol. 27, no. 3, pp. 267 - 289, 2006.

[18] Y. Gu, . A. McCallum and D. Towsley, "Detecting Anomalies in Network Traffic Using Maximum Entropy Estimation," *The 5th ACM SIGCOMM Conference on Internet Measurement*, Berkeley, CA, USA, pp. 32 - 32, 2005 .

[19] Y. Zheng, Y. Liu, . Y. Jing and X. Xie, "Urban Computing with Taxicabs," *The 13th International Conference on Ubiquitous Computing*, Beijing, China, pp. 89 – 98, 2011.

[20] J. Yang and L. Meng, "Feature Selection in Conditional Random Fields for Map Matching of GPS Trajectories," Journal *on Location-Based Services*, Springer International Publishing, 2014, pp. 121-135.

[21] S. Liu, Y. Liu and L. M. Ni, "Towards Mobility-based Clustering," *The 16th ACM SIGKDD International Conference on* Knowledge *Discovery and Data Mining*, Washington, DC, USA, pp. 919 – 928, 2010.

[22] J.G. Lee, J. Han and K.Y. Whang, "Trajectory Clustering: A Partition-and-Group

Framework," *The 2007 ACM SIGMOD International Conference on Management of Data*, Beijing, China, pp. 593 – 604, 2007.

[23] Y. Zheng, Y. Liu, J. Yuan, and X. Xie, “Urban Computing with Taxicabs,” *The 13th International Conference on Ubiquitous Computing*, Beijing, China, pp. 89 – 98, 2011.

[24] H. W. Chang, Y. C. Tai, H. W. Chen and J. Y. j. Hsu, "iTaxi: Context-Aware Taxi Demand Hotspots Prediction Using Ontology and Data Mining Approaches," *Intelligent Transportation Systems* , Florence, Italy, 2008.

[25] X. Li, G. Pan, Z. Wu, G. Qi, S. Li, D. Zhang, W. Zhang and Z. Wang, "Prediction of Urban Human Mobility Using Large-Scale Taxi Traces and Its Applications," *Computer Science*, vol. 6, no. 1, pp. 111 - 121, 2012.

[26] B. Li, D. Zhang, L. Sun, C. Chen, S. Li, G. Qi and Q. Yang, "Hunting or Waiting? Discovering Passenger-Finding Strategies from a Large-Scale Real-World Taxi dataset," *Pervasive Computing and Communications Workshops*, Seattle, WA, USA, 2011.

[27] H. Hu, Z. Wu, B. Mao, Y. Zhuang, J. Cao and J. Pan, "Pick-Up Tree Based Route Recommendation from Taxi Trajectories," *The 13th International Conference on Web-Age Information Management*, Harbin, China, 2012.

[28] Q. Zou, G. Xue, Y. Luo, J. Yu and H. Zhu, "A Novel Taxi Dispatch System for Smart City," *Journal of Distributed, Ambient, and Pervasive Interactions,* 2013.

[29] Y. Shen, L. Zhao and J. Fan, "Analysis and Visualization for Hot Spot Based

RouteRecommendation Using Short-Dated Taxi GPS Traces," *Journal in Information* , vol. 6, pp. 134 - 151, 2015.

[30] C. E. Leiserson, C. Stein, R. Rivest and T. H. Cormen, "Introduction to Algorithms," MIT Press ed., USA, 1990.

[31] P. Hart, N. Nilssonx and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics,* vol. 4, no. 2, p. 100 –107, 1968.

[32] Y. Lim and H. Kim, "A Shortest Path Algorithm for Real Road Network Based on Path Overlap," *Journal of the Eastern Asia Society for Transportation Studies*, vol. 6, pp. 1426 -1438, 2005.

[33] A. Awasthi, M. Parent and P. J.M., "Rule Based Prediction of Fastest Paths on Urban Networks," *Intelligent Transportation Systems*, Bertini, 2005.

[34] J. Yuan, Y. Zhen, L. Zhang, X. Xie and G. Sun, "Where to Find My Next Passenger?," *The 13th International Conference on Ubiquitous Computing*, Beijing , China, pp. 109 – 118, 2011.

[35] Gonzalez, H. J and X. Li, "Adaptive Fastest Path Computation on a Road Network: A Traffic Mining Approach," *The 33rd International Conference on Very large Data Bases*, Vienna, Austria, pp. 794 – 805, 2007.

[36] Jula, M. Dessouky and P. A. Ioannou, "Real-Time Estimation of Travel Times Along the Arcs and Arrival Times at the Nodes of Dynamic Stochastic Networks," *IEEE Trans. on Intelligent Transportation Systems*, vol. 9, no. 1, 2008.

[37] G. Linden, B. Smith and J. York, "Amazon.com Recommendations," *the IEEE Computer Society*, 2003.

[38] D. Pfoser, . C. S. Jensen and . Y. Theodoridis, "Novel Approaches to the Indexing of Moving Object Trajectories," *The 26th International Conference on Very Large Data Bases*, Cairo, Egypt, pp. 395 – 406, 2000.

[39] I. Trestian, K. Huguenin, L. Su and K. Aleksandar, "Understanding Human Movement Semantics: A Point of Interest Based Approach," *The* 21st International World Wide Web Conference (WWW2012), New York, USA, pp. 619 – 620, 2011.

[40] M. Vlachos, G. Kollios and D. Gunopulos, "Discovering Similar Multidimensional Trajectories," *The 18th International Conference on Data Engineering*, San Jose, CA, USA, pp. 673 – 684, 2002.

[41] S. Ghemawat, H. Gobioff and S.T. Leung, "The google file system," *The 19th ACM Symposium on Operating Systems Principles*, New York, 2003.

[42] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM - 50th Anniversary Issue: 1958 - 2008*, vol. 51, no. 1, pp. 107-113, 2008.

[43] H.C. Yang, A. Dasdan, R.L. Hsiao and D. S. Parker, "Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters," *The 2007 ACM SIGMOD international conference on Management of Data*, New York, pp. 1029- 1040, 2007.

[44] M. Isard, M. Budiu, Y. Yu, A. Birrell and D. Fetterly, "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks," *The 2nd ACM*

SIGOPS/EuroSys European Conference on Computer Systems, Lisbon, Portugal, pp. 59 – 72, 2007.

[45] A. Gates, O. Natkovich, S. Chopra, P. Kamath, S. Narayanam, C.Olston, B. Reed, S. Srinivasan and U. Srivastava, "Building a HighLevel Dataflow System on top of MapReduce: The Pig Experience," *Very Large Data Bases,* vol. 2, no. 2, pp. 1414 – 1425, 2009.

[46] P. P. Anchalia, A. K. Koundinya and S. N, "MapReduce Design of K-Means Clustering Algorithm," T*he International Conference on Information Science and Applications*, Suwon, pp. 1 – 5, 2013.

[47] F. Sun, W. Wang, B. Zhou and F. Chen, "The Design and Application of Navigation and Location Services Data Index," *The 5th International Conference on Computational and Information Sciences*, Wuhan, China, pp. 774 – 777, 2013.

[48] Q. Ma, B. Yang and W. Qian, "Query Processing of Massive Trajectory Data based on MapReduce," *The 1st International Workshop on Cloud Data Management*, Hong Kong, China, pp. 9 – 16, 2009.

[49] Y. Fang, R. Cheng, W. Tang, S. Maniu and X. Yang, "Evaluating Nearest-Neighbor Joins on Big Trajectory Data," *Tech. Report,* Hong Kong, China, 2014.

[50] A. Eldawy and M. F. Mokbel, "A Demonstration of SpatialHadoop: An Efficient MapReduce Framework for Spatial Data," *Journal on the VLDB Endowment*, Riva del Garda, Trento, Italy, 2013.

[51] D. Li, X. Zhou, Q. Wang and M. Gao, "Map-Reduce for Calibrating Massive Bus

Trajectory Data," *The 13th International Conference on ITS Telecommunications*, Washington, DC, USA, pp. 44 – 49, 2013.

[52] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the ACM*, pp. 509-517, 1975.

[53] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *The 1984 ACM SIGMOD International Conference on Management of Data*, USA, pp. 47-57, 1984.

[54] A. Labrinidis and H. V. Jagadish, "Challenges and opportunities with big data," 2012. [Online]. Available: http://www.cra.org/ccc/files/docs/init/bigdatawhitepaper.pdf.

[55] S. M. Omohundro, "Five Balltree Construction Algorithms," *ICSI Technical Report* , 1989.

[56] N. Roussopoulos, S. Kelly and F. Vincent, "Nearest neighbor queries," *The 1995 ACM SIGMOD International Conference on Management of Data*, USA, pp. 71-79, 1995.

[57] D. Li, X. Zhou, Q. Wang and M. Gao, "Map-Reduce for Calibrating Massive Bus Trajectory Data," *The 13th International Conference on ITS Telecommunications*, Bristol, United Kingdom, 2013.

[58] L. Liu, C. Andris and C. Ratti, "Uncovering cabdrivers' behavior patterns," *Journal on Computers, Environment and Urban Systems*, no. 34, pp. 541 - 548, 2010.

[59] L. Liu, A. Biderman and C. Ratti, " Urban Mobility Landscape: Real Time

Monitoring of Urban Mobility Patterns," *The 11th International Conference on Computers in Urban Planning and Urban Management*, Hong Kong, 2009.

[60] W. Liu, Y. Zheng and S. Chawla ,"Discovering Spatio Temporal Causal Interactions in Traffic Data Streams," *The 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, San Diego, California, USA, 2011.

[61] T. Hunter, R. Herring , P. Abbeel and A. Bayen, "Path and Travel Time Inference from GPS Probe Vehicle Data," *Workshop on NIPS Analyzing Networks and Learning with Graphs*, New York, USA, 2009.

[62] K. Yamamoto, K. Uesugi, T. Watanabe, "Adaptive Routing of Cruising Taxis by Mutual Exchange of Pathways," *International Journal of Knowledge Engineering and Soft Data Paradigms*, Zagreb, Croatia, 2008.

[63] J.W. Powell1, Y. Huang, F. Bastani and M. Ji, "Towards Reducing Taxi Time Using Spatio-Temporal Profitability Maps," *The 12th International Symposium*, Minneapolis, MN, USA., 2011.

[64] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," *The 29th International Conference on Data Engineering*, Brisbane, Australia, pp. 410 – 421, 2013.

[65] F. Pai, "A hybrid ARIMA and support vector machines model in stock price forecasting," *The International journal of Management Science*, p. 497–505, 2005.

[66] O. Wolfson, X. B, C. S and . L. Jiang, "Moving objects databases: Issues and solution," *M. Rafanelli & M. Jarke, eds, 'SSDBM',IEEE Computer Society,* pp. 111 -

122, 1998.

[67] J.G. Lee, J. Han, X. Li and H. Gonzalez, "TraClass: Trajectory Classification Using Hierarchical Region-Based and Trajectory-Based Clustering," *The VLDB Endowment*, Auckland, New Zealand, pp. 1081 - 1094, 2008.

[68] L. X. Pang, S. Chawla and W. Liu, "On Mining Anomalous Patterns in Road Traffic," *The 2011 International Conference on Advanced Data Mining and Applications*, Beijing, China, 2011.

[69] G. Gidofalvi and X. Huang, "Probabilistic Grid-Based Approaches for Privacy Preserving Data Mining on Moving Object Trajectories," *Journal on Privacy-Aware Knowledge Discovery,* Strasbourg, pp. 183–210, 2010.

[70] N. S. Savage, S. Nishimura and . N. E. Chavez, "Frequent Trajectory Mining on GPS Data," *The 3rd International Workshop on Location and the Web,* Tokyo, Japan, 2010.

[71] X. Li, J. Han, J.-G. Lee and H. Gonzalez, "Traffic Density-Based Discovery of Hot Routes in Road Networks," Boston, MA, USA, 2007.

[72] Y. Yue, Y. Zhuang, Q. Li and Q. Mao, "Mining Time-dependent Attractive Areas and Movement Patterns from Taxi Trajectory Data," *The 10th International Conference on Advances in Spatial and Temporal Databases,* Fairfax, Virginia, USA, pp. 441 – 459, 2009.

[73] S. Proper and P. Tadepalli, "Solving Multiagent Assignment Markov Decision Processes," *The 8th International Conference on Autonomous Agents and Multiagent*

Systems, Budapest, Hungary, pp. 681 – 688, 2009.

[74] Lu, C. Lin and . V.Tseng, "Mining the Shortest Path within a Travel Time Constraint in Road Network Environments," *The 11th International IEEE Conference on Intelligent Transportation Systems*, Beijing, China, pp.593 – 598, 2008.

[75] Z. Gui, H. Yu and Y. Tang, "Locating Traffic Hot Routes from Massive Taxi Tracks in Clusters," *Novel Applications and New Technique,* 2013.

[76] B. Yang, Q. Ma, W. Qian and A. Zhou, "TRUSTER: TRajectory Data Processing on ClUSTERs," *The 14th International Conference on Database Systems for Advanced Applications*, Brisbane, Australia, 2009.

[77] a. Seki, R. Jinno and K. Uehara, "Parallel Distributed Trajectory Pattern Mining Using Hierarchical Grid with MapReduce," *International Journal of Grid and High Performance Computing (IJGHPC),* vol. 5, no. 4, pp. 79 - 96, 2013.

[78] D. Zhang and T. He, "pCruise: Reducing Cruising Miles for Taxicab Networks," *The 33rd Conference on Real-Time Systems Symposium*, San Jan, pp.85 – 94, 2012.

[79] Z. Liao, "Taxi Dispatching via Global Positioning Systems," *Journal of Engineering Management,* vol. 48, no. 3, pp. 342 - 347, 2011.

[80] D. Lee and H. Wang, "A Taxi Dispatch System based on Current Demands and Real-Time Traffic," *Transportation Research Record,* vol. 1882, pp. 193 - 200, 2004.

[81] L. Tang and X. Chang, "Public Travel Route Optimization Based on Ant Colony Optimization," *China Journal of Highway and Transport,* vol. 24, no. 2, pp. 89 - 95,

2001.

[82] X. Zhou, H. Zhao and J. Yu, "Taxi Calling and Scheduling System Based on GPS," *Journal of Computer Engineering and Design*, vol. 30, no. 21, pp. 4995 - 4997, 2009.

[83] E. Kanoulas, Y. Du and D. Z. Tian Xia, "Finding Fastest Paths on A Road Network with Speed Patterns," *The 22nd International Conference on Data Engineering*, Atlanta, GA, pp. 10 -10, 2006.

[84] S. Bekhor and M. S. Ramming, "Evaluation of Choice Set Generation Algorithms for Route Choice Models," *Annals of Operations Research*, vol. 144, no. 1, pp. 235 - 247, 2006.

[85] J. H, W. Cao, J. Luo and X. Yu, "Dynamic Modeling of Urban Population Travel Behavior based on Data Fusion of Mobile Phone Positioning Data and FCD," The 17th International Conference on Geoinformatics, Fairfax, Virginia, USA, pp. 1- 5, 2009.

[86] B. Gacias and F. Meunier, "Design and operation for an Electric Taxi Fleet," *Journal of OR Spectrum. Quantitative Approaches in Management*, vol. 37, pp. 171- 194, 2015.

[87] S.F. Cheng and X. Qu, "A Service Choice Model for Optimizing Taxi Service Delivery," *The 12th International Conference on Intelligent Transportation Systems*, St. Louis, USA, pp. 1- 6, 2009.

APPENDIX: PUBLICATION DURING THE PROGRAM

Wenxin Yang, Xin Wang, Mohammadreza Rahimi Seyyed, Jun Luo, 2015: *Recommending Profitable Taxi Travel Routes based on Big Taxi Trajectories Data*, The 19th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2015), Ho Chi Minh City, Vietnam.