Traffic Prediction and Navigation Using Historical and Current Information

by

Sejoon Lim

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2008

Author ....
Department of Electrical Engineering
and Computer Science
February 1, 2008

Certified by ....
Daniela Rus
Professor
Thesis Supervisor

Accepted by ....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Traffic Prediction and Navigation Using Historical and Current Information

by

Sejoon Lim

Submitted to the Department of Electrical Engineering
and Computer Science
on February 1, 2008, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering
and Computer Science

Abstract

We developed a traffic prediction and navigation system that deals with uncertainty of road traffic conditions by stochastic modeling of road networks. Our system consists of a data collecting system, a data management system, and a path planning system. First, the data collecting system gathers real-time travel time data using a mobile sensor network system, CarTel. GPS sensor units having wireless connectivity were deployed on taxis running around the Boston area, and report their position and time information to the networked database system. Second, the raw GPS data collected from this CarTel system is processed to generate a database storing the statistical information of road travel time. We organize a large amount of data in a form in which they can be accessed efficiently and can capture important aspects of road traffic conditions. Third, we developed efficient stochastic shortest path algorithms that find best paths depending on drivers' goals. We evaluate our algorithms using both simulations and real-world drives. Finally, we implemented a path planning system using historical and current information organized by our data management system. Our system provides a Web-based interface that is publicly usable. The interface provides traffic information, including optimal paths and visualized traffic conditions. Our system also offers analysis tools of users' own driving routes with user track-log uploading interface. We evaluate the system using taxi trajectories and human driving experiments.

Thesis Supervisor: Daniela Rus
Title: Professor

Acknowledgments

I would like to express my gratitude to Daniela Rus for her support and supervision. I would not have been able to finish my Master's work without her guidance and help. Her vision and enthusiasm for this research were really helpful to develop our project. I am also very grateful to Hari Balakrishnan, David Gifford, and Samuel Madden for their suggestions and guidance. I appreciate Jakob Eriksson for providing me with the CarTel GPS data. Thanks to all the members of the Distributed Robotics Laboratory for helpful discussions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Traffic congestion is clearly a serious problem: a recent survey [36] estimates that the annual nation-wide cost of traffic congestion is $78 billion, including 4.2 billion hours in lost time and 2.9 billion gallons in wasted fuel. Drivers today have little knowledge of historic and real-time traffic congestion on the paths they drive, and even when they do (e.g., from "live" traffic updates), they generally do not know how to use that information to find good paths. As a result, they often tend to drive sub-optimal routes and often leave well in advance when they need to make an important deadline.

Our objective is to provide an effective navigation system for cars that uses historical and real-time traffic data to determine optimal driving directions and traffic estimates. Our work provides a planning system that can be used by robots as well as human drivers. The system is a useful addition to on-board navigation systems using computer-aided automation to provide good paths that meet desired travel goals (e.g., "when should you leave, and what path should you take, to reach the airport by 8am with high probability?"); it is also a worthwhile addition to Web-based mapping services. We view the incorporation of traffic-aware path computation as an important practical addition in the rapid trend toward computer-assisted driving and autonomous decision-making in vehicles.

In addition to helping individual cars avoid congested roads, we believe that our work, if deployed widely, can manage traffic flow, reduce congestion, and reduce the fuel consumed by cars on a macroscopic basis by using the under-utilized parts of the road network better than today (thereby reducing load on congested areas). Using our algorithm to investigate this global traffic management question is an area for future work. In this thesis, we are concerned with finding good paths for a single car.

We present and evaluate algorithms for planning the motion of vehicles (autonomous or human-driven) on roadways in the face of traffic delays. Rather than model road delays statically, as in current on-board navigation systems and Web-based mapping services, our algorithm uses past observations of actual delays on road segments to model these delays as probability distributions. The algorithm minimizes a user-specified cost function of the delay distribution. We investigate a few cost functions in detail, particularly one that is equivalent to maximizing the likelihood of reaching a destination within a specified travel deadline.

The main challenge in planning paths taking traffic delay into account is that these delays are not fixed. The delay on a road segment is best modeled as a probability distribution; in addition, this distribution typically depends on a number of factors, such as time-of-day, whether it is a working day or not, events such as concerts or sporting events, weather, etc. The shortest-distance path is often not the best path to use if one seeks to minimize the expected travel time or maximize the probability of reaching the destination by a certain time. Our algorithm uses historic observations of travel delays on road segments at different times of day to produce delay distributions (indexed by time-of-day). We posit that this information, together with real-time updates of extraneous conditions (such as accidents), is invaluable (and sufficient) to compute good paths that meet user-specified goals. Given the probability distributions of delays on segments, finding good paths requires more than a shortest path computation, because the "optimal substructure" property does not hold (i.e., if the best path from S to T goes through X, it does not follow that the sub-path of this path from S to X is itself the best S-X path).

We have implemented our algorithm and evaluate it by first modeling the historic delays using data from the CarTel vehicular testbed [21], a network of 28 taxis. The data consists of travel times organized by road segment and by time of day, yielding statistical profiles for all the road segments. We model the road network as a weighted graph where the nodes represent intersections and the edges represent road segments. An aggregation algorithm combines the road segments into groups to coalesce the important delay characteristics without losing information about alternate paths. Our algorithm has the flavor of searching and pruning the delay statistics on the road network data structure. We evaluate the algorithm and its assumptions using simulation and actual test driving.

Observation of traffic conditions is essential for traffic prediction and navigation. There have been various approaches to measuring road speed or flow, from using sensors buried under roads to using cameras. However, methods utilizing these kinds of equipment are expensive to establish. Thus, it may be impossible to obtain live traffic information for all the roads. In contrast, our approach involves acquiring traffic data from various sources that are already established and wide-ranging, such as GPS sensors, and records of toll booth pass times. As a result, we may be able to estimate the speed of entire roadways without any deployment of additional hardware systems on roads.

Even if we knew the current road velocity of all the roads, several challenges would remain. One problem is that the best path based on the current information might not actually be optimal since the road conditions could change while we travel. What we really need to know is the velocity of each road at the time we actually drive through it. In fact, it would be impossible to tell the traffic conditions exactly since traffic conditions might change, but we envision that we can predict the future conditions of roads by examining how the past and current traffic conditions affect the future condition. Thus, we use not only real-time traffic information but also historical information to develop a probabilistic model for predicting traffic conditions.

Another challenge is finding an optimal path when road travel times are uncertain. For example, if you want to go from your home to the airport tomorrow morning such that you reach the airport by 8am, what is the latest time you can leave home and what route should you use, such that you make it by that time with high probability? Our system allows users to obtain answers to such questions. There have been some stochastic shortest path algorithms in theoretical area, but practical algorithms are rare. In addition, most previous research assumes that the travel times for road sections follow the Gaussian distribution and that they are independent of each other, but none of these algorithms has been supported by any concrete experiments. In our research, we investigated whether these assumptions are realistic by examining real data.

Because we deal with a large amount of data, it is essential to organize data in an efficient way. And fast implementation is needed. These are non-trivial issues.

To achieve our objective addressing the technical challenges, first, we collected real-time travel time data by a networked sensor network with GPS sensors. Second, we organized the data to extract important traffic condition information from the raw GPS measurement data. We observed that the traffic pattern has both uncertainty and predictability. To capture these, we use stochastic modeling of road network travel time. We used them for predicting traffic conditions, travel time estimates and for finding the best route. Finally, we developed efficient algorithms for finding the optimal paths with stochastic conditions. We want to provide drivers information which route would be the best for the drivers, using the information measured by our network sensors. The notion of "best" route depends on the drivers' goals. Typically, goals can be expressed as combinations of speed and reliability. Based on the developed algorithms, we designed and implemented a traffic prediction and navigation system.

Contributions of our work include:

Real-time wide-range high-granularity data collection. Networked cars gather and deliver travel time data for every road segment where cars reach in real time. The data is wide ranging since GPS sensors can report their position and time everywhere cars go. The data has high-granularity since cars can go every small road segment. The data was gathered over a long period with a large amount of cars that run many hours per day. Our database system is unique in the abundance of real measurement data.

Statistical modeling and building a road travel time map. The road network was modeled as statistical delay map. A robust map matching alg orithm, a road segments combining algorithm for meaningful delay statistics, and a current information propagation algorithm were developed to build a travel time map, which is a basis of traffic prediction and route planning.

Efficient stochastic shortest path algorithms. Stochastic shortest path algorithms were developed, implemented, and evaluated. The algorithms build upon some previous work [32, 29] on stochastic shortest path algorithms and improves the performance.

Publicly usable traffic navigation system. We implemented the algorithms making a working Web-based traffic navigation system. The system is accessible through internet via Web browsers.

Interface for gathering public drivers' track-log data. We get public users' GPS traces to enlarge our GPS log database. We encourage their participation by providing analysis tools for their driving habits.

1.1 Outline

This thesis is organized as follows. Chapter 3 gives an overview of a route planning system for traffic. Chapter 4 gives the description of the data collection system. Chapter 5 gives the data processing and analysis module. Chapter 6 gives the stochastic motion planning formulation, presents our algorithms, and evaluates the algorithms in simulation and also using physical data from the CarTel taxi deployment. Chapter 7 gives a method of using sparse current information. Chapter 8 gives our traffic information system. Chapter 9 gives the field experiment to test our system.

Chapter 2

Related Work

This thesis builds on very important previous work in motion planning, data collection and analysis, and traffic prediction systems. Specifically, we extend the algorithmic work in [32] who introduced a very nice algorithm for stochastic motion planning.

2.1 Shortest Path Problem

Since efficient algorithms for shortest path problems were proposed by Bellman [17] and Dijkstra [16], many algorithms have been developed for finding the shortest paths in various networks. A^* search algorithm [25, 15] finds the shortest path using a heuristic estimate, which is used to rank the paths to search. These algorithms work if the edge weights are deterministic and do not change. In many applications of road travel planning, however, the more important measure of the path optimality is the travel time, which changes according to various factors depending on networks. In [13], an efficient algorithm for a dynamic shortest path with time dependent deterministic edge weights is given. In [22], the authors suggested an efficient query method that finds the set of all fastest paths, one for each sub-interval having different edge travel times, given a user-defined leaving or arrival time interval. These works consider the time dependency of the travel time, but do not consider the uncertainty involved in the traffic.

Several prior studies have considered the stochastic nature of the travel time [18, 38, 9, 27, 41, 29, 31, 32]. In the stochastic shortest path problem, edge weights are modeled as probability distributions rather than deterministic values. In this setting, the optimal path depends on drivers' diverse objectives. When a driver's objective is to minimize the expected travel time, the problem can be solved by the standard shortest path algorithms by replacing all the edge weights with their expected values as is first approached in [18]. However, for various goals such as maximizing the probability of arriving within a given deadline, the optimal path cannot be found with the standard shortest path algorithms since the optimal substructure property does not hold. In [18], the author examined some methods to find the shortest-path probability distributions by approximation using Monte Carlo programming.

Loui [28] suggested using a general non-decreasing monotonic utility function of arrival time to represent the goodness of reaching the destination at a certain time and gave an algorithm that finds the path that maximizes the expected value of a utility function. However, it yields a large running time. Nikolova et al. [32] developed an algorithm that finds the path that maximizes the probability of reaching the destination in a given deadline assuming that delays are both Gaussian and independent on different road segments. Inspired by this algorithm, we developed a method that improves performance by removing unnecessary invocations of shortest path searches. Murthy and Sarkar [29] developed a stochastic shortest path algorithm with a decreasing deadline utility function based on pruning paths using dominance conditions. We developed a second method that improves performance by clarifying the pruning conditions when the edge cost follows Gaussian distributions and the utility function is a simple deadline function.

2.2 Data Sources

There have been several approaches to acquiring traffic data. The most prevalent one uses inductor loops installed beneath roads [43, 14]. This is adequate for counting the number of cars that pass a specific location, but it is not suitable for measuring travel time, and measurements are possible only on instrumented roads. Recently, GPS sensors installed in probe vehicles have been used [35, 21, 45]. The travel time of vehicles can be measured and recorded for each road segment. In [21], the researchers developed a system called CarTel that includes GPS and wireless communication. This system was used to study routing and data delivery from cars. Recently, some companies [4, 1] use cell phone localization to estimate the road traffic conditions. They track the cell phone movement to get the velocity of roads or measure the volume of cell phones to estimate the amount of congestion. Their service focuses on visualizing the level of congestion on the map. Alternative paths provided by the systems would be helpful for avoiding congestion but may be sub-optimal depending on users' various goals.

2.3 Map Matching

To use the GPS data as a source of traffic measurement, the GPS location should be associated with the underlying map. This process is known as map matching. Map matching methods vary from simple search techniques [23], to more complex techniques such as using Kalman filters or Belief Theory [24, 39, 37, 30]. The methods using statistical estimation work well especially when the GPS signal can be lost for a certain amount of time, but usually require heavy calculations. In [8, 42] the point to point, point to arc, and arc to arc matchings are introduced, but they are unlikely to work very well unless both the user's location and the map are known with a high degree of accuracy. Matching GPS points onto a map requires smart techniques since GPS sample points have noise and maps are not perfectly aligned to the real road segments in many cases. The authors in [8, 42, 20, 34] suggest map matching algorithms that utilize the network topological information to improve the map matching performance. In [40] vehicle speed was taken into account to compliment inaccurate vehicle heading information at low speeds. In [10], the authors suggested global map matching using the Fréchet Distance as a distance measure. The algorithm considers only minimizing the maximum distance between GPS points and a candidate path, which could lead to mismatching in case some GPS samples have large noise. Most of the previous research focused on identifying the correct link among the candidate links in every step of the map matching process since one bad match can lead to a sequence of bad matches. However, none of them effectively deals with the accumulated map-matching errors. We suggest an algorithm that detects and recovers the map matching errors by post-processing.

2.4 Traffic Prediction

In predicting traffic conditions, it has usually been assumed that traffic patterns repeat from day to day, and that changes in traffic happen gradually. Many existing models for predicting traffic conditions use historical traffic information, real-time traffic information, or both [7, 44, 26]. In [14], the authors employed an artificial neural network technique for traffic forecasting, where long-term forecasts rely on historical patterns, and short-term forecasts rely on current information. A spatial extrapolation method was used to reconstruct the present traffic situation from point measurements [11]. In [19] the authors used a belief propagation algorithm to predict traffic conditions as a binary status indicating traffic on a segment is fluid or congested. They used floating car data sent by probe vehicles as the only source of traffic information. There has been also some research on traffic prediction based on simulation considering various factors that affect the traffic flow [33, 7]. In the literature, traffic prediction focuses on predicting a future condition as a specific value, but in this thesis we will be more interested in estimating the distributions of the travel time in the road networks.

Chapter 3

High Level System Description

Our research objective is to provide an effective navigation system for autonomous or human-piloted cars that uses historical and real-time traffic data to determine optimal driving directions and traffic estimates. Our intelligent navigation system consists of:

1. a data gathering system (e.g. CarTel nodes [21]) included in cars that move in traffic frequently (e.g. taxis);
2. a data analysis system to compile a historical database of traffic conditions;
3. an algorithm for route planning that uses both historical data and current information;
4. a traffic information system implementation with an appropriate human interface.

The first part of the system consists of data gathering systems to obtain traffic delay data. In [21] a system called CarTel was developed that uses GPS and wireless communication to collect position and time data from cars. CarTel is a mobile sensor network; each car carries a wireless embedded computer and a collection of sensors, including GPS. The result is network capable of obtaining data about a large metropolitan area at relatively low cost, compared to current approaches that deploy traffic sensors in roads. CarTel nodes deployed in 28 Boston taxis since January 2007 collected many gigabytes of traffic delay information over the past year, covering several thousands of driving routes.

The second part of the system converts this data into a statistical model of traffic delays segmented according to various temporal and spatial categories. This task is challenging

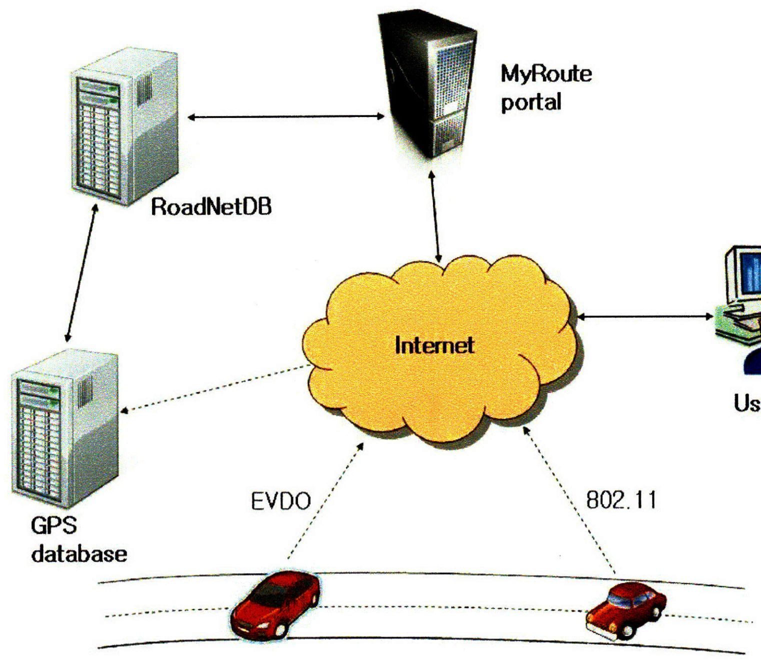


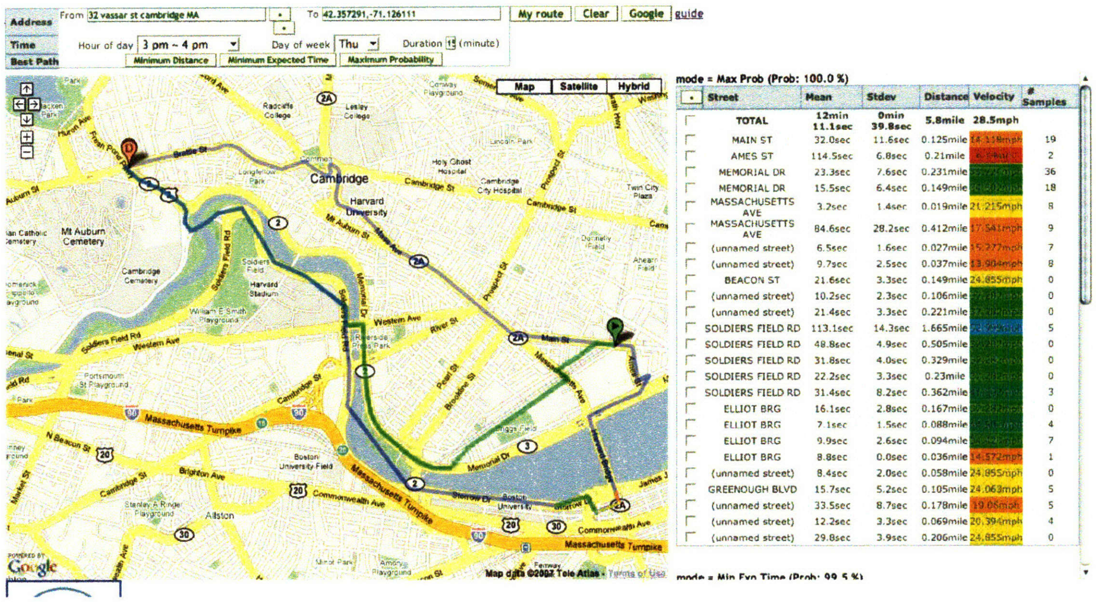
Figure 3-1: System Overview. The data gathering systems are composed of networked cars with GPS sensors and the GPS database where location and time data of each car is stored. The raw GPS data is processed and stored in an organized form in the database RoadNetDB. MyRoute portal Web service provides users traffic information including best route and travel time estimates, etc.

for several reasons: first, data volumes are quite high, so efficient algorithms and indices are needed. Second, the data is noisy, so algorithms that match GPS points to an underlying road database in a robust and error free way are essential, as also discussed in prior work by other researchers. Third, determining how to segment the data into a collection of road segments is non-trivial; simply computing statistics on a per-intersection granularity leads to a huge road graph with questionable statistics for each segment. However, making the segments too large loses information about alternate path segments to route cars along.

The third part of the system uses these statistical distributions of road delays to produce delay- and congestion-aware paths. We have developed new algorithms that optimize various delay criteria, building on recent theoretical work on stochastic shortest paths algorithms [32].

A Web-based interface allows users to query the system for traffic conditions and for optimal paths given historical data. (see `http://gasherbrum5.csail.mit.edu:`

```
8000/cgi-bin/trafficDemo.py)
```



Address From 32 vassar st cambridge MA To 42.357291,-71.126111

Time Hour of day 3 pm ~ 4 pm Day of week Thu Duration 15 (minute)

Best Path

mode = Max Prob (Prob: 100.0 %)

Street	Mean	Stdev	Distance	Velocity	# Samples
TOTAL	12min 11.1sec	0min 39.8sec	5.8mile	28.5mph	
MAIN ST	32.0sec	11.6sec	0.125mile	14.118mph	19
AMES ST	114.5sec	6.8sec	0.21mile	[illegible]	2
MEMORIAL DR	23.3sec	7.6sec	0.231mile	[illegible]	36
MEMORIAL DR	15.5sec	6.4sec	0.149mile	[illegible]	18
MASSACHUSETTS AVE	3.2sec	1.4sec	0.019mile	21.215mph	8
MASSACHUSETTS AVE	84.6sec	28.2sec	0.412mile	17.541mph	9
(unnamed street)	6.5sec	1.6sec	0.027mile	15.277mph	7
(unnamed street)	9.7sec	2.5sec	0.037mile	13.904mph	8
BEACON ST	21.6sec	3.3sec	0.149mile	24.855mph	0
(unnamed street)	10.2sec	2.3sec	0.106mile	[illegible]	0
(unnamed street)	21.4sec	3.3sec	0.221mile	[illegible]	0
SOLDIERS FIELD RD	113.1sec	14.3sec	1.665mile	[illegible]	5
SOLDIERS FIELD RD	48.8sec	4.9sec	0.505mile	[illegible]	0
SOLDIERS FIELD RD	31.8sec	4.0sec	0.329mile	[illegible]	0
SOLDIERS FIELD RD	22.2sec	3.3sec	0.23mile	[illegible]	0
SOLDIERS FIELD RD	31.4sec	8.2sec	0.362mile	[illegible]	3
ELLIOT BRG	16.1sec	2.8sec	0.167mile	[illegible]	0
ELLIOT BRG	7.1sec	1.5sec	0.088mile	[illegible]	4
ELLIOT BRG	9.9sec	2.6sec	0.094mile	[illegible]	7
ELLIOT BRG	8.8sec	0.0sec	0.036mile	14.572mph	1
(unnamed street)	8.4sec	2.0sec	0.058mile	24.855mph	0
GREENOUGH BLVD	15.7sec	5.2sec	0.105mile	24.063mph	5
(unnamed street)	33.5sec	8.7sec	0.178mile	19.06mph	5
(unnamed street)	12.2sec	3.3sec	0.069mile	20.394mph	4
(unnamed street)	29.8sec	3.9sec	0.206mile	24.855mph	0

mode = Min Exp Time (Prob: 99.5 %)

Figure 3-2: Demonstration Web Interface. The figure shows different optimal paths from a start location (the green arrow) to a destination (the red "D) on a Thursday afternoon, according to the three different criteria: the route recommended by Google's Maps (the topmost route), minimum expected time (the middle route), and maximum probability of arriving by the deadline (the bottom route).

This Web interface allows users to select start point, end point, hour of day, day of week, and deadline, and provides the user with the minimum distance path, the minimum expected time path, and the maximum arrival probability path.

Chapter 4

Data

In this chapter we discuss the collection of traffic data using CarTel nodes, and the integration of this data with existing traffic databases.

4.1 CarTel System

Real-time traffic data is collected by a wireless sensor network built by CarTel [21] (see `http://cartel.csail.mit.edu/`). CarTel is a mobile sensor computing system designed to collect, process, deliver, and visualize data from sensors located on mobile units such as automobiles. It provides a simple, centralized programming interface, handles large volumes of sensor data, and copes with variable, intermittent network connectivity. A CarTel node is a mobile embedded computer. Each node gathers and processes sensor readings locally before delivering them to a central database. A variety of on-board and external sensors collect data as users drive. The possible applications of this mobile sensor computing system includes traffic monitoring by using GPS sensors, environmental monitoring by using chemical and pollution sensors, civil infrastructure monitoring such as patholes by using vibration sensors, automotive diagnostics by obtaining information from a vehicles's on-board sensors, geo-imaging by taking location-tagged images, and data muling by using cars as delivery networks.

4.2 GPS Data Collected

Cars with the CarTel node (Figure 4-1) collect data as they drive and log them to their local databases. As connectivity to any open wireless access point via 802.11 protocol becomes available, data on cars is delivered to the CarTel GPS database server, which is PostgreSQL 8.0.7. Some CarTel GPS units also have Evolution-Data Optimized (EVDO) connection so that they are guaranteed to report their location in real time. 28 taxis in the Boston area are equipped with the EVDO enabled CarTel nodes and are reporting their positions every second in real time when they are running. The data covers anywhere taxis run. The data fields stored in the CarTel GPS database are *latitude*, *longitude*, *datetime*, and *label of the car*.

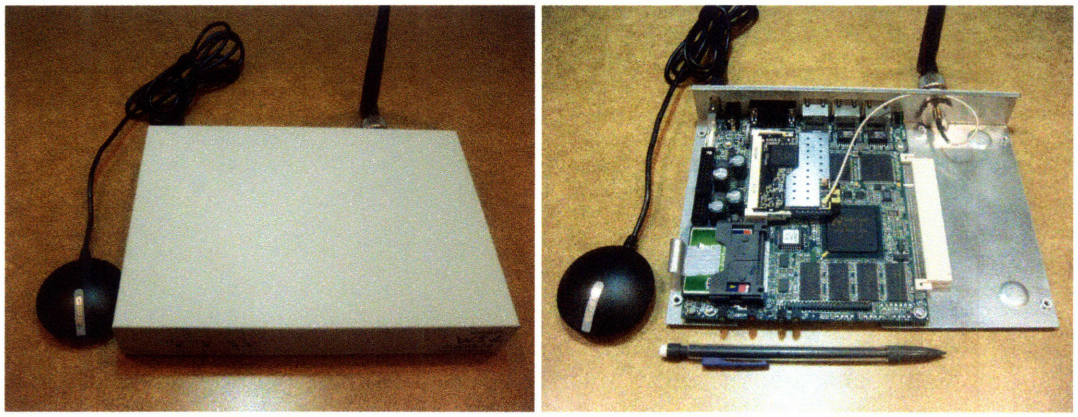


Figure 4-1: CarTel node hardware [21]. Soekris net4801 has a 586-class processor running at 266 MHz with 128 MB of RAM, 1GByte of flash memory, an 802.11b miniPCI Wi-Fi card, and a USB port through which GPS sensors are connected.

The total recorded time by 28 taxis from February to November 2007 is 80506350 seconds, which corresponds to about 932 days. The number of trips is 68515. Thus, the average trip time is about 19 minutes 35 seconds. Each car drove an average of about 47 days for this period, which means that each taxi gathered data for about 3.7 hours a day. The car who gathered the most data drove 8039127 seconds (equivalent to about 93 days), which corresponds to average 7.4 hours a day.

Figure 4-2 shows how the collected GPS samples are distributed spatially. From the plot we can see that the data is quite abundant for the Boston and Cambridge area. Even though the density of data for suburban areas is not high we can see that the range of the data gathered is quite wide.

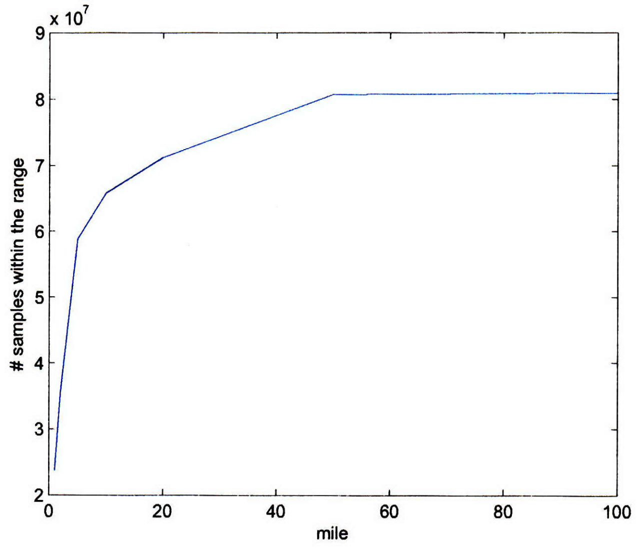


Figure 4-2: Cumulative number of GPS samples according to the range. Y-axis indicates the number of GPS samples within the distance indicated in X-axis from MIT

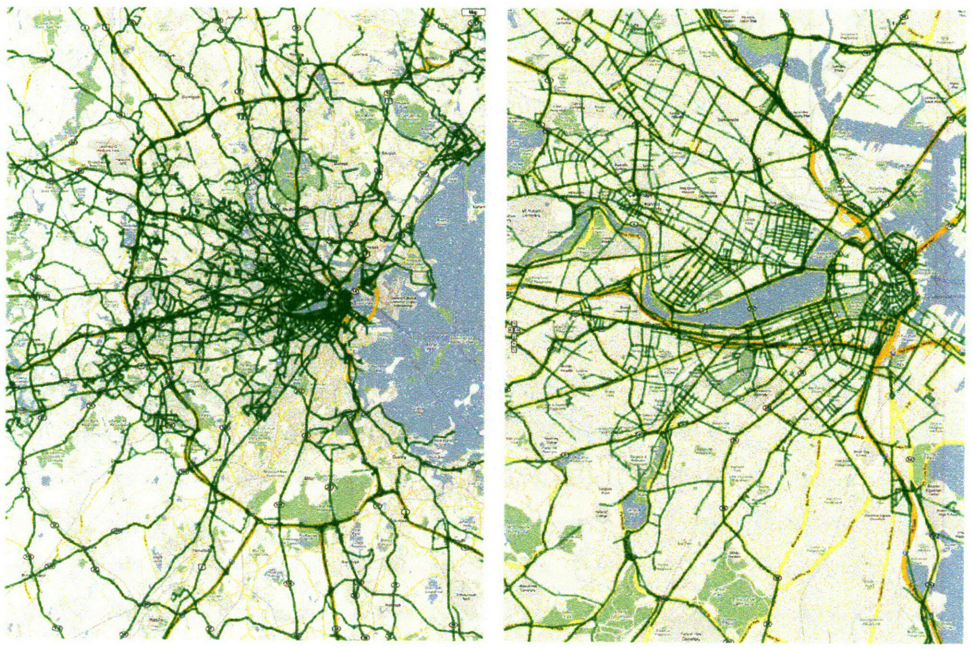


Figure 4-3: The Road Segments of Massachusetts Frequently Covered by the Taxis with the GPS Nodes. The right figure is a zoomed-in version of the left figure.

Figures 4-3 and 4-4 shows the road segments covered by the 28 taxies. The figures were generated after matching the GPS data onto a map by using the map-matching algorithm that is described later in this thesis.

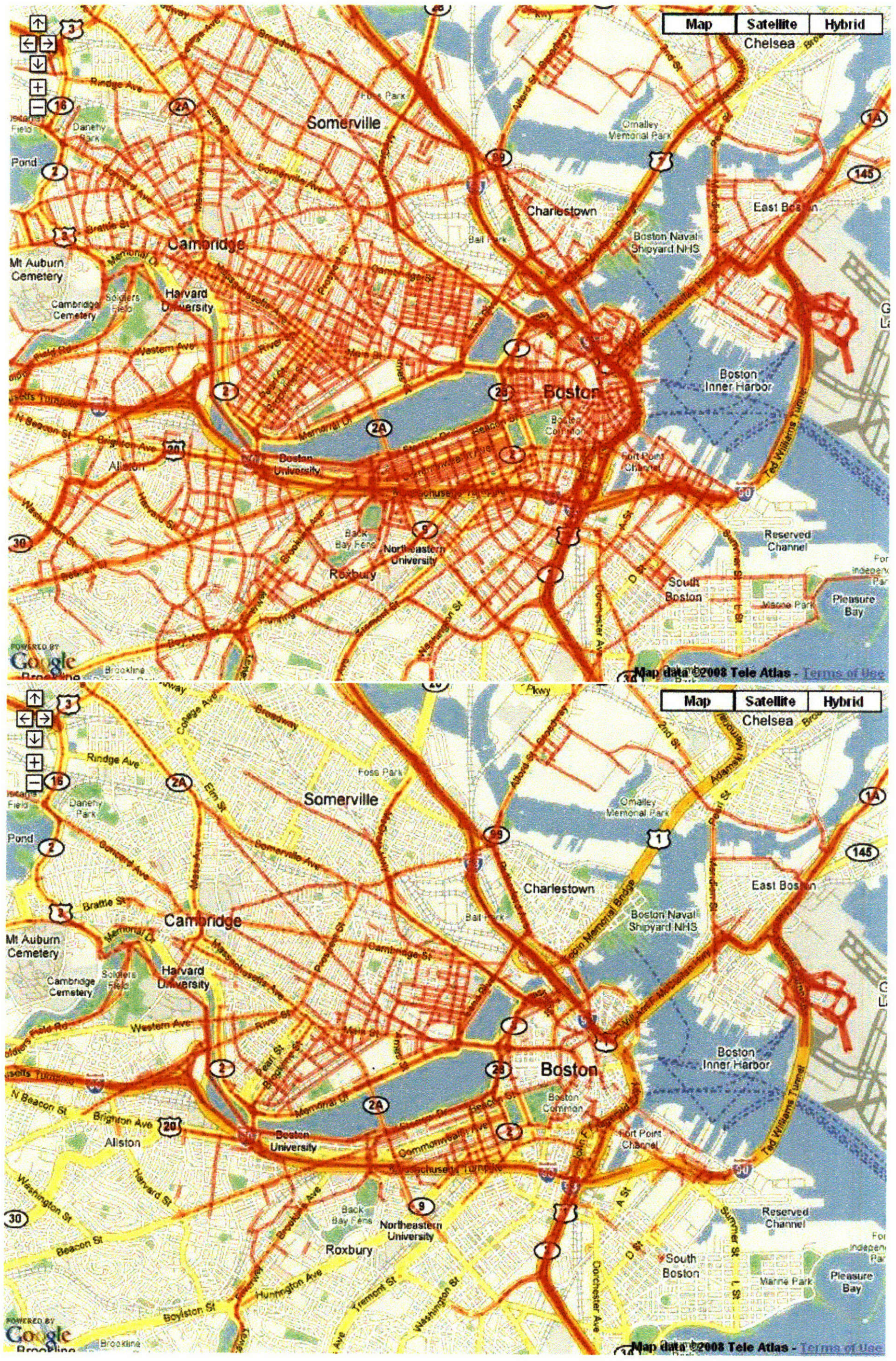


Figure 4-4: The Road Segments of Boston and Cambridge Covered by the Taxis with the GPS Nodes. Road segments where the number of coverage is more than 10 times (top) and road segments where the number of coverage is more than 100 times (bottom).

4.3 Map Data

4.3.1 TIGER/Line Data

The TIGER/Line files are extracts of selected geographic and cartographic information from the Census Bureau's TIGER (Topologically Integrated Geographic Encoding and Referencing system) database [6]. The TIGER/Line files are publicly available. The files contain rows representing each road segment. Fields in multiple columns describe the features of each road segment, such as latitude and longitude of the two end points, street name, street category, etc. Figure 4-6 shows the road segments of the TIGER/Line files on the Google Maps [3]. The road network built from the TIGER/Line files for Massachusetts has 429,529 segments and 341,773 intersections.

As shown in Figure 4-6, the TIGER/Line files describe every small road segments with relatively high accuracy. However, there are some drawbacks that it difficult to use the TIGER/Line data for routing purpose. First, there is no one-way information. Second, there is no restricted maneuver information, such as whether left turn is prohibited or not at a junction. Third, there is no multi layer information. Consider a two layer crossing as illustrated in Figure 4-5. Let the two thick arrowed lines represent a portion of highway and the thin line represent a surface road beneath the highway. Thus, the three lines are not physically connected, but the TIGER/Line files describe this multi-layer road structure as 7 road segments, a$\sim$g, where each road segment is denoted by its two end points with just latitude and longitude. So, there is no way to distinguish this kind of multi layer crossing with 7 physically connected road segments.

Other issues that make routing with the TIGER/Line files difficult include that the map is not well-aligned to the real road segments in some areas and that no speed limit or average speed information exists.

4.3.2 Navteq Data

Navteq is a digital map company whose products are widely used in many commercial GPS navigation systems [5]. We obtained Navteq Sample data for the northeast region of the

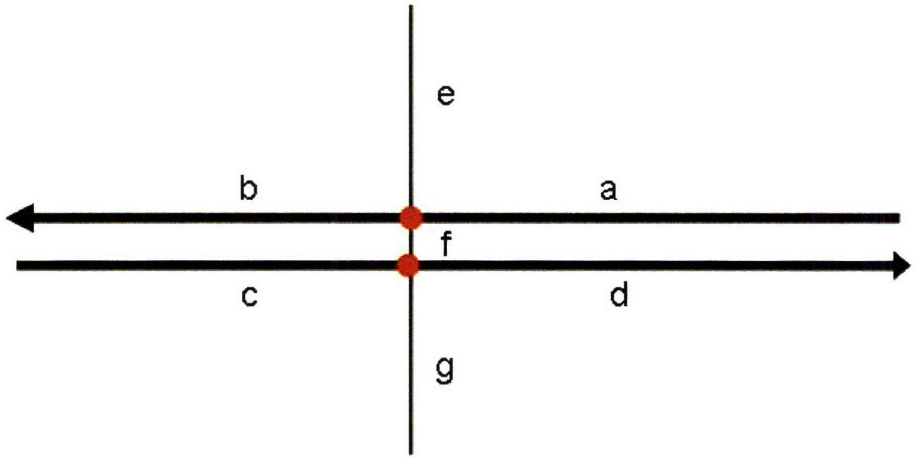


Figure 4-5: The Representation of Multi-level Crossing

United States in the GDF (Geographic Data Files) format [2]. The Navteq data contains useful information for routing including one-way, restricted maneuver, multi-layer information. The road network build from the Navteq data for Massachusetts, Rhode Island, and Connecticut has 602,268 segments and 468,366 intersections. The Navteq road segments around the Boston and Cambridge area are shown in Figure 4-7. Comparing Figure 4-6 and Figure 4-7, we observe that the Navteq data is better aligned to the real road segments than TIGER/Line data.

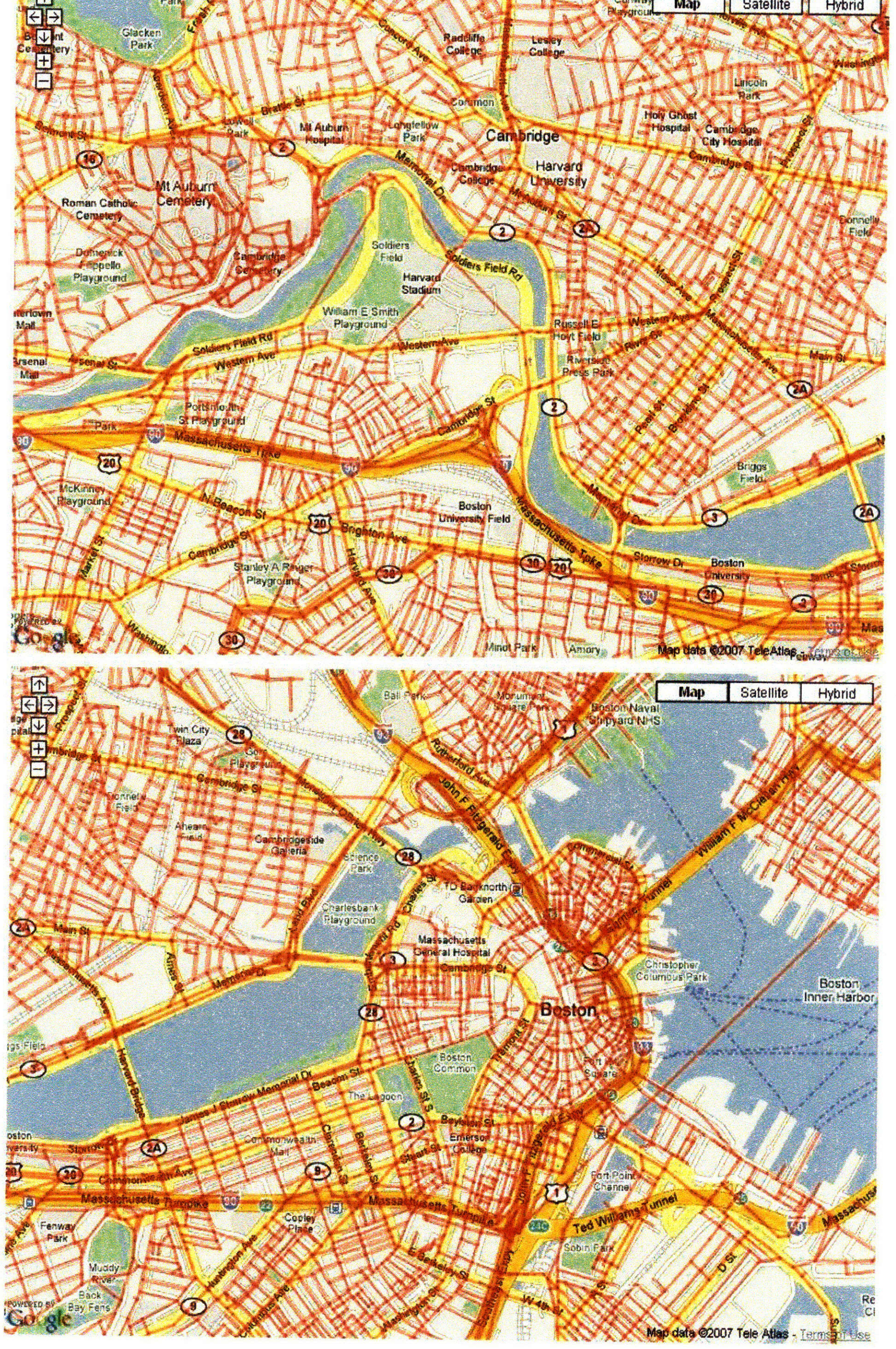


Figure 4-6: Road Segments in the TIGER/Line Data around Boston (Top) and Cambridge (Bottom)

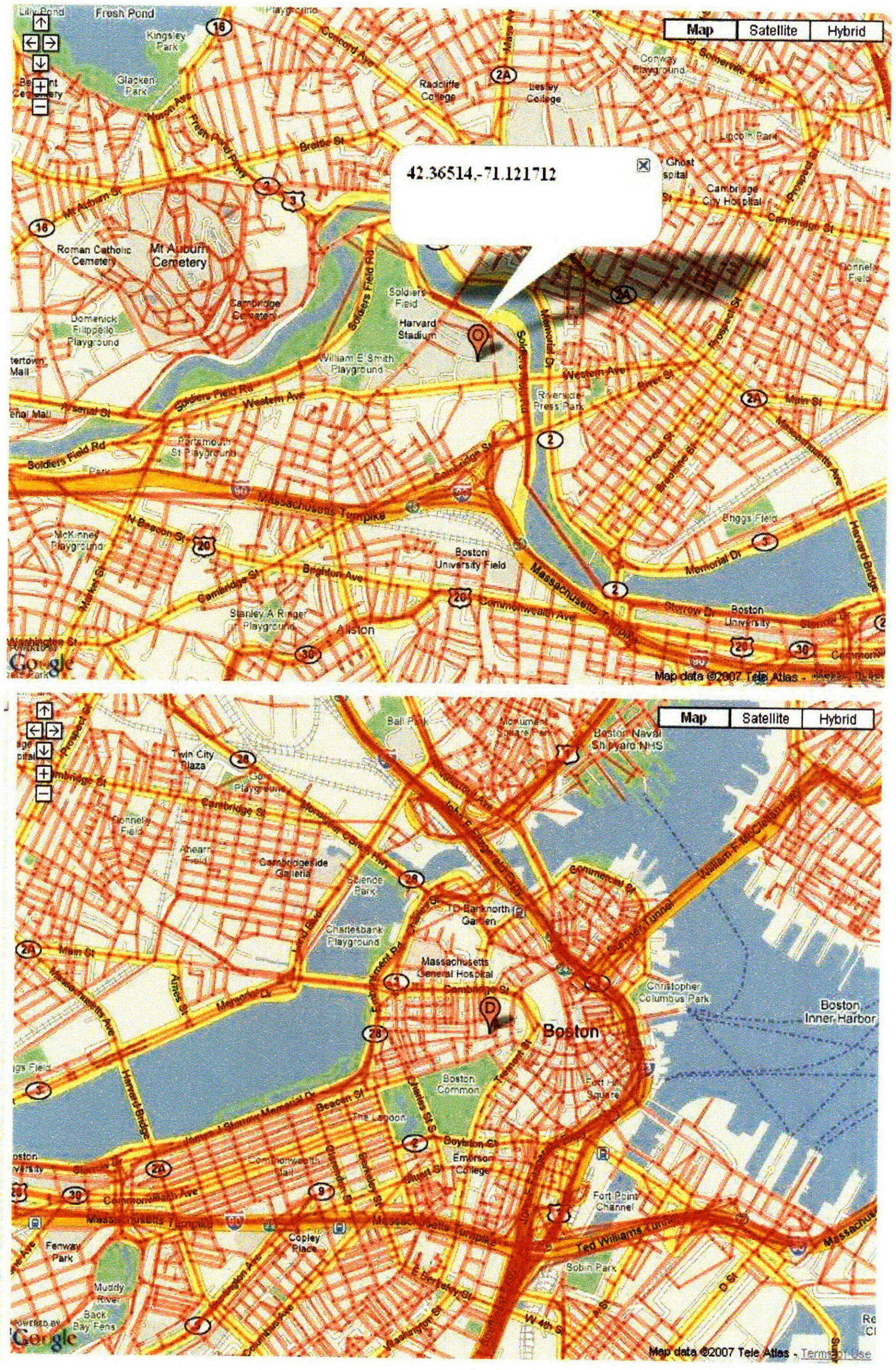


Figure 4-7: Road Segments in the Navteq Sample Data around Boston (Top) and Cambridge (Bottom)

Chapter 5

Data Processing

In this chapter we describe how we organize traffic information and store it in our database.

5.1 Road Network Modeling

The road network is represented by a graph, where nodes represent intersections and edges represent road segments. We call this the *Geographic Map*. We associate a road delay distribution with each road segment. This per-intersection granularity road map leads to a large graph for small road segments with related travel statistics. We combine statistically related road segments into groups so that they can capture important delay characteristics without losing information about alternate path segments. This data structure is the *Delay Statistics Map*. The Geographic Map is used for matching CarTel GPS traces onto real road segments, while the Delay Statistics Map is used for statistical-delay-sensitive routing.

5.2 Matching GPS Data onto a Map

Since the coordinates of points in map databases are not always well-aligned to real road segments, and GPS measurement data are noisy, we need to have a robust map matching algorithm. Because of these errors, finding the best road segments corresponding to a few GPS samples does not work well. To solve this, we developed a post-processing algorithm that can be used to correct local matching errors.

Our map matching algorithm uses forward local matching and backward post processing. First, GPS trajectory samples are grouped by five points (line 4 of Algorithm 1). Then, the most probable road segment where the five points might have originated is selected (line 5 of Algorithm 1). For this local matching, the distance and orientation between each road segment and every five points are considered (line 7 of Algorithm 2). Noting that the maximum vehicle speed is around 100 miles per hour and that the minimum road segment size is about 50 meters, the sampling rate of our GPS tracking unit (one sample per second) implies that at least one sample will lie on each road segment through which probe vehicles actually pass. Thus, it is guaranteed that GPS sample points cover connected sets of road segments. The post processing is based on this observation (line 6 of Algorithm 1). If a new candidate road segment is not connected to the previous segment (line 2 of Algorithm 3), we select the next best candidate (line 17 of Algorithm 3). If this candidate is not also connected, we examine a maximum of three more candidates until we achieve connectivity. If connectivity is satisfied within these five trials, the post-processing ends and the found segment is inserted into the segment list (line 7 of Algorithm 1) and we proceed with the local matching with the next five sample points. If we encounter a situation where we cannot find connectivity with the best five candidates (line 5 of Algorithm 3), we consider that local matching has failed at one of the previous steps. Then, we increase the aging factor of the previous road segment by a factor of 2 and go one step back and find the possible maximum five candidates (line 6 - 11 of Algorithm 3). The cost of the local matching is multiplied by this aging factor (line 10 of Algorithm 2). If we find any candidate in 5 trials, then we proceed with the local matching keeping the found candidate segment. Otherwise, we go another step back, and do the same procedure. With this recursive post processing algorithm, we achieve robust map matching in the presence of GPS measurement noise and digital map misalignment.

Figure 5-1 shows the comparison of map matching between the case when we use the local-matching as well as post-processing and the case when we only use the local-matching. The benefit of post-processing is remarkable especially when GPS location errors are large or the underlying map is not accurate.

Driver's stop detection. Taxi drivers make intentional stops regardless of road traffic

Algorithm 1: MAP-MATCHING

```
Data: GPS trajectory $t$
Result: sequence of segments $segList$, which corresponds to the input trajectory $t$
1 $segList \leftarrow []$
2 $agingIDs \leftarrow []$: aging id list
3 $agingCounts \leftarrow []$: aging count list corresponding to the aging id list
4 foreach 5 sample subtrajectory s in t, which is shifted by one sample do
5     $bestSeg \leftarrow$ LOCAL-MATCHING($s, agingIDs, agingCounts$)
6     POST-PROCESSING($bestSeg, segList, agingIDs, agingCounts$)
7     $segList.append(bestSeg)$
8 return $segList$
```

Algorithm 2: LOCAL-MATCHING

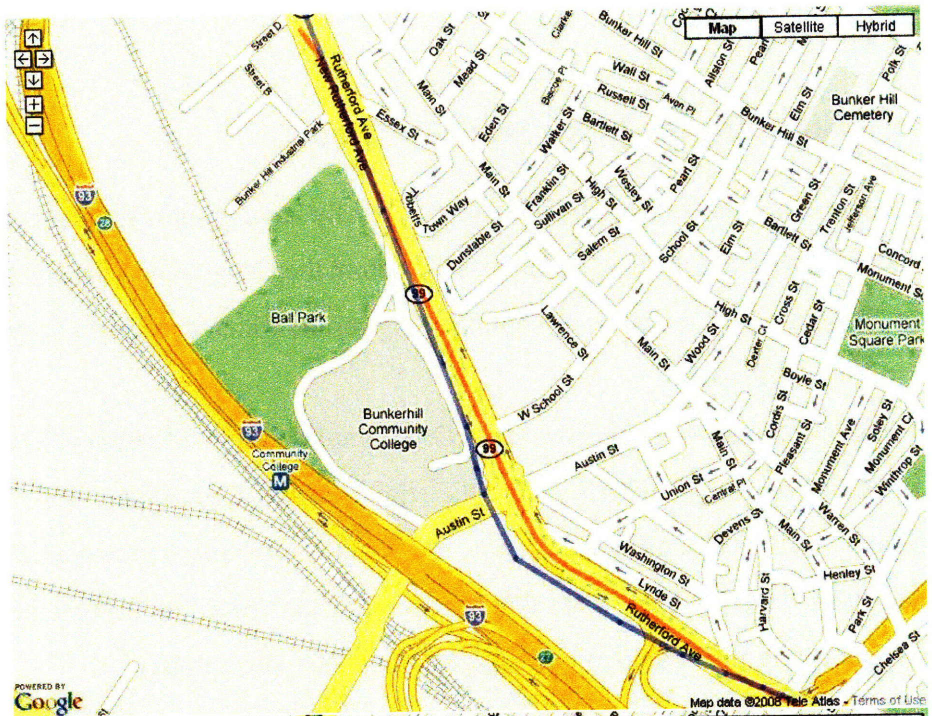
```
Data: subtrajectory $s$, aging id list $agingIDs$, aging count list $agingCounts$
Result: the local best segment $bestSeg$
1 $bestSeg \leftarrow \varnothing$
2 $minCost \leftarrow 0$
3 $costFactor \leftarrow 1$: the factor multiplied to the actual cost according to aging
4 foreach $point \in s$ do
5     $streetSet.add(getClosestStreets(point))$
6 foreach $seg \in streetSet$ do
7     $cost \leftarrow meanDistance(s, seg) + varDistance(s, seg) + orientation(s, seg)$
8     if $seg.id \in agingIDs$ then
9         $costFactor \leftarrow 2 * agingCounts$
10    $cost \leftarrow costFactor * cost$
11    if $cost < minCost$ then
12        $bestSeg \leftarrow seg$
13        $minCost \leftarrow cost$
14 return $bestSeg$
```

conditions. Thus, we need to classify when the taxi drivers stopped due to traffic and when they stopped due to other needs, and discard the waiting times not due to traffic. If the distance of the two end points of 2 minute trajectory is less than 3 meters (which means less than 0.1 km / hour) then we consider it as a stop by a driver. This is from frequent observation that it is very rare that a car would not move even 3 meters for 2 minutes even though the roads were congested. If we make the threshold too small, e.g., less than 3 meters, GPS noise becomes dominant. More accurate models that analyze more long-term patterns of a car movement would improve the stop detection.

Algorithm 3: POST-PROCESSING

```
Data: $bestSeg$, $segList$, $agingIDs$, $agingCounts$
Result: modified $agingIDs$, $agingCounts$
1  $lastSeg \leftarrow$ the last element in $segList$
2  while $bestSeg$ is not connected to $lastSeg$
3  and $bestSeg$ is not the same with $lastSeg$ do
4      $localMatchingFailCount \leftarrow localMatchingFailCount + 1$
5      if $localMatchingFailCount > 5$ then
6          remove the last element of $segList$ and roll back the subtrajectory pointer to
           the start point of $lastSeg$
7          if $lastSeg.id \in agingIDs$ then
8              increase the $agingCounts$ for the $lastSeg$ by 1
9          else
10             $agingIDs.append(lastSeg.id)$
11         $lastSeg \leftarrow$ the last element in $segList$
12     if $bestSeg.id \in agingIDs$ then
13         increase the $agingCounts$ for the $bestSeg$ by 1
14     else
15         $agingIDs.append(bestSeg.id)$
16     $s \leftarrow 5$ sample subtrajectory starting from the modified subtrajectory pointer
17     $bestSeg \leftarrow$LOCAL-MATCHING$(s, agingIDs, agingCounts)$
18 return $agingIDs$, $agingCounts$
```

Map matching failure detection. In some cases, map matching fails. In such cases we need to discard these samples. After map matching, we compare the actual distance of the trajectories and the distance sum of matched segments. If the difference between them is larger than a predetermined threshold then we classify this as a map-matching failure.



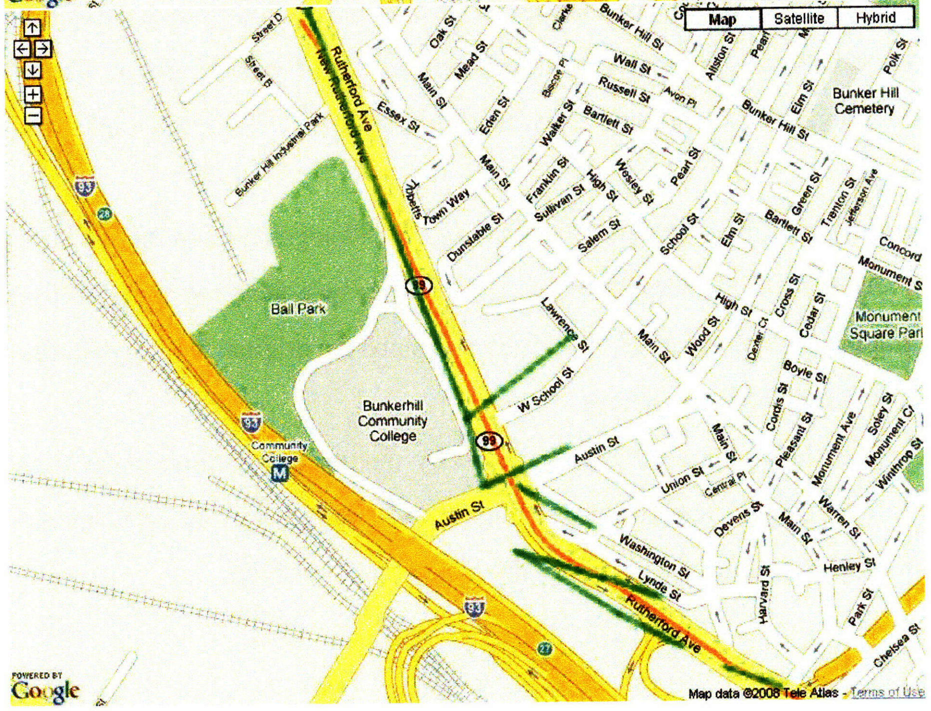


Figure 5-1: Map Matching Result. The red line is actual GPS points reported from the CarTel node and blue line is a sequence of segments that were matched by Algorithm 1 (left). The green line is a sequence of segments that were selected without post-processing (right).

5.3 Aggregation

5.3.1 Constructing the Delay Statistics Map

We need to combine the road segments into meaningfully large groups. In this section, we will explain how we build the Delay Statistics Map from the Geographic Map and the *Map Matching Table*, a table created as a map matching result, which contains a sequence of rows for matched segments.

After map matching, we know how many GPS samples each road segment has. Using this information, we make a graph, called the *High Samples Geographic Map*, that has only segments with enough samples. That is, low sample segments are removed from the Geographic Map. In this graph, nodes that have more than three edges become junctions. Segments connecting any couple of junctions are combined into a group.

The following is the detailed road segments combining algorithm that assigns a unique groupID and direction compared to the group direction for each group. The root node should be a node that connects at least three segments in the High Samples Geographic Map.

We generated a Delay Statistics Map having 38,844 segments and 28,698 intersections. This is a reduction of the number of nodes and edges by a factor of about 20 from that of the Geographic Map represented by Navteq data since the original Navteq data has 602,268 segments and 468,366 intersections.

Figure 5-2 shows the road network built by this process, which is the graph representation of the Delay Statistics Map. Figure 5-3 shows what the GPS sample number distribution is like among the groups of the Delay Statistics Map.

```
Algorithm 4: BUILD THE DELAY STATISTICS MAP
Data: High Samples Geographic Map table
Result: $Group$ table representing Delay Statistics Map
$groupIDs[-1, -1, ...]$: size=number of segments
$directions[0, 0, ...]$: size=number of segments
$node \leftarrow rootNode$
$nodes$ : FIFO queue
$newGroup \leftarrow True$
$curGroupID \leftarrow 0$
while $node \neq \varnothing$ do
    if $node \neq rootNode$ then
        if $node$ was already visited then
            $node \leftarrow nodes.pop()$
        else
            $groupIDs[node.segmentID] \leftarrow curGroupID$
            $directions[node.segmentID] \leftarrow node.direction$
            $groupDistance \leftarrow groupDistance + node.distance$
            if $newGroup$ then
                $newGroup \leftarrow False$
                $startNode \leftarrow node.prev$
    $neighbors \leftarrow getNodeNeighborsOnHighSamplesGeographicMap(node)$
    foreach $neighbor \in neighbors$ do
        if $neighbor$ was not visited then
            $nodes.append(neighbor)$
    if $len(neighbors) \neq 2$ then
        $endNode \leftarrow node$
        if $curGroupID \neq 0$ then
            $insertIntoDatabaseGroupTable($
            $curGroupID, startNode, endNode, groupDistance)$
        $curGroupID \leftarrow curGroupID + 1$
        $newGroup \leftarrow True$
        $groupDistance \leftarrow 0$
    $node \leftarrow nodes.pop()$
```

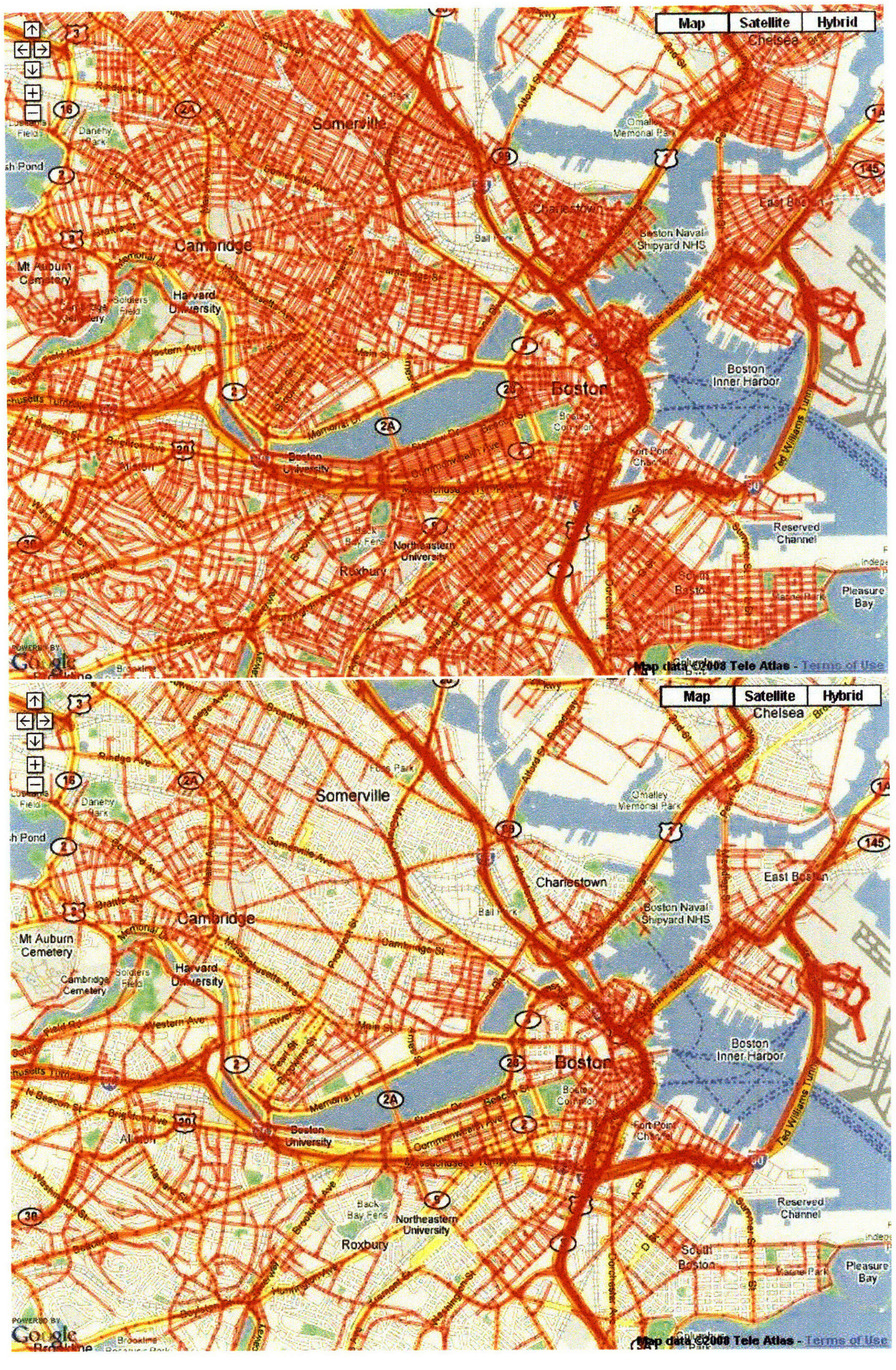


Figure 5-2: The Geographic Map (top) and the Delay Statistics Map Built by Algorithm 4 (bottom)

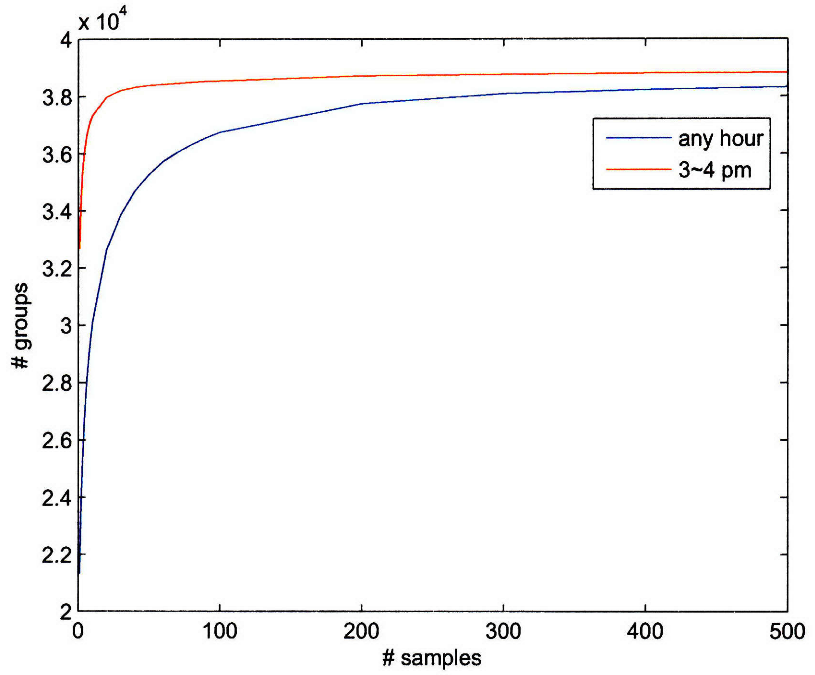


Figure 5-3: The Number of Groups That Have At Most The Amount of Samples Indicated in X-axis. There are total 38844 groups. The maximum number of samples on a group is 10917.

5.3.2 Assigning Travel Time Statistics

Scanning the Map Matching Table we assign the mean and variance of each group by the method described in Algorithm 5. For each segment in the Map Matching Table (line 4), we accumulate the segment delay as long as they belong to the same group (lines 6-9), and update the group statistics by the process described in lines 11-18.

Algorithm 5: ASSIGN DELAY STATISTICS

```
   Data: Map Matching Table $mmt$, $Groups$ tables
   Result: Updated mean, variance, numSampls fields in $Groups$ tables
1  $groupDelay \leftarrow 0$
2  $prevGroupID \leftarrow -1$
3  $first \leftarrow True$
4  foreach $row \in mmt$ do
5      $(groupID, delay) \leftarrow groupIDandDelays(row)$
6      if $first$ or $groupID == prevGroupID$ then
7          $groupDelay \leftarrow groupDelay + delay$
8          $prevGroupID \leftarrow groupID$
9          $first \leftarrow False$
10     else
11         $(numSamples, meanTime, varTime, distance) \leftarrow$
               $getCurrentStatistics(prevGroupID)$
12         $newNumSamples \leftarrow numSamples + 1$
13         $newMeanTime \leftarrow$
               $(meanTime * numSamples + groupDelay)/newNumSamples$
14         $newVarTime \leftarrow (numSamples * (meanTime^2 + varTime) +$
               $groupDelay^2)/newNumSamples - newMeanTime^2$
15         $setNewStatistics(prevGroupID, newNumSamples, newMeanTime,$
16             $newVarTime)$
17         $groupDelay \leftarrow delay$
18         $prevGroupID \leftarrow groupID$
```

Chapter 6

Stochastic Planning

In this chapter we describe our two novel algorithms that cope with the uncertainty of road traffic conditions by stochastic modeling of travel delay on road networks. These algorithms determine paths between two points that optimize a cost function of the delay probability distribution. They can be used to find paths that maximize the probability of reaching a destination within a particular travel deadline. For such problems, standard shortest path algorithms do not work because the optimal substructure property does not hold. In the worst case, stochastic shortest path problems are computationally intractable, requiring the exploration of a search space that is super-exponential in the number of road segments. A suite of stochastic route planning algorithms was developed and implemented. The best route depends on the drivers' goals and is a combination of speed and reliability. Our algorithm can be integrated into on-board navigation systems as well as route-finding Web sites, providing good paths to drivers that meet their desired goals.

6.1 Problem Formulation

6.1.1 Road Network Modeling

A road network is represented by a graph, where edges represent road segments and nodes represent intersections. We assume that

1. the delay of each edge follows a Gaussian distribution;

2. the delay of each edge is independent of every other edge.

In Section 6.4, we provide evidence for these assumptions. We formulate stochastic motion planning as a graph search problem over a graph with an origin O and a destination D, where the travel time of each edge is an independent Gaussian random variable. Since the sum of independent Gaussian random variables is also a Gaussian random variable, we can denote the travel time for a path π consisting of edges e of mean m_e and variance v_e as follows:

$$t_\pi \sim \mathcal{N}(m_\pi, v_\pi), \text{ where } m_\pi = \sum_{e \in \pi} m_e \text{ and } v_\pi = \sum_{e \in \pi} v_e.$$

6.1.2 Cost Functions

Our objective is to find a path that minimizes an expected cost when the cost function models a user's goal. We call this the "optimal" path for the given cost function. We consider several cost functions including:

Linear Cost

Here, the cost increases linearly with the travel time. When the cost of arriving at the destination in time t is $C(t) = t$ and the delay PDF of a path π is $f_\pi(t)$, the expected cost of traveling through π is

$$EC_\pi = \int_{-\infty}^{\infty} t f_\pi(t) dt = m_\pi.$$

Linear cost models the path with minimum expected time.

Exponential Cost

Exponential cost models a cost function that increases sharply as the arrival time increases. When the cost function is $C(t) = e^{\lambda t}$, where λ is the steepness of the cost increase, the expected cost can be written as

$$EC_\pi = \int_{-\infty}^{\infty} e^{\lambda t} f_\pi(t)dt = \{e^{\lambda(m_\pi + \frac{\lambda v_\pi}{2})}\}.$$

This exponential cost function minimizes a linear combination of mean and variance determined by λ.

Step Cost

Step cost models a cost that only penalizes the late arrival after a given deadline. The cost function is $C(t, d) = u(t - d)$, where $u(\cdot)$ is the unit step function and d is the deadline. The expected cost is

$$\begin{aligned} EC_\pi(d) &= \int_{-\infty}^{\infty} u(t-d) f_\pi(t)dt = \int_{d}^{\infty} f_\pi(t)dt \\ &= \{1 - \Phi(\frac{d - m_\pi}{\sqrt{v_\pi}})\}, \end{aligned}$$

where $\Phi(\cdot)$ is the CDF of the Standard Normal distribution. Thus, when Π is a set of all paths from O to D, the minimum expected cost path is

$$\underset{\pi \in \Pi}{\operatorname{argmax}} \ \Phi(\frac{d - m_\pi}{\sqrt{v_\pi}}), \tag{6.1}$$

which turns out to be the path that maximizes arrival probability. Since $\Phi(\cdot)$ is monotonically increasing, maximizing $\Phi(\cdot)$ in (6.1) is equivalent to maximizing

$$\varphi_d(\pi) = \frac{d - m_\pi}{\sqrt{v_\pi}}. \tag{6.2}$$

The minimum expected cost path for the linear and exponential cost cases can be found by a deterministic shortest path algorithm, such as Dijkstra's algorithm or A^* search algorithm since the cost of a path can be expressed as the sum of the cost of each edge in the path. However, when the cost is a step function, these algorithms cannot be used since the objective, (6.2), is nonlinear. Our goal for the rest of the thesis is to develop an efficient algorithm for finding the maximum probability path given a deadline.

6.2 Stochastic Path Planning by Parametric Optimization

In [32], an algorithm for the case of normally distributed edge costs was given based on quasi-convex maximization. It finds the path with the maximum arrival probability by standard shortest path runs with different edge costs corresponding to varying parameters. We now give a graphical interpretation of the optimal path and show a connection to the parametric optimization problem, which will ultimately lead to a new algorithm that reduces unnecessary runs over [32].

6.2.1 Transforming the Cost Function Into Parametric form

Let path π be denoted by a point (m_π, v_π) in a rectangular coordinate system, called the $m - v$ plane, where the horizontal axis represents the mean and the vertical axis represents the variance. The objective of the optimization problem, (6.2), can be rewritten to show the relation between m_π and v_π as

$$v_\pi = \frac{1}{\varphi_d(\pi)^2}(m_\pi - d)^2, \tag{6.3}$$

which is a parabola in the $m - v$ plane with apex at d, where $\varphi_d(\pi)$ is determined by the curvature of the parabola. Thus, the optimal path is the path that lies on the parabola of the smallest curvature.

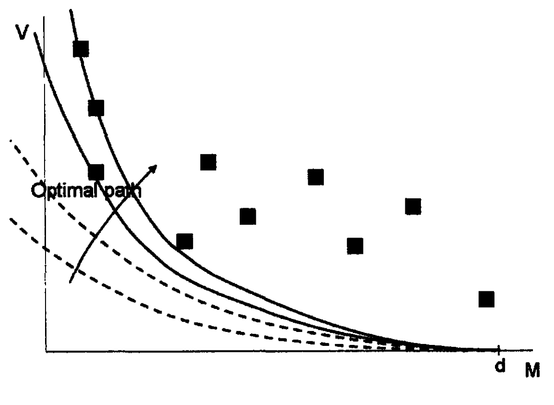


Figure 6-1: Graphical Interpretation of the Optimal Path in the $m - v$ Plane. Each black square represents a path from the origin to the destination. Equi-probability paths lie on a parabola with an apex at $(d, 0)$ and a curvature of $\frac{1}{\varphi_d(\pi)^2}$. The optimal path is the first point that meets with a parabola as we increase the curvature.

Intuitively, the optimization problem is to find the first path that intersects the parabola while we lift up the parabola starting from the horizontal line (see Figure 6-1). This suggests finding the optimal path using linear optimization with various combinations of cost coefficients.[1]

Consider setting the cost of an edge be linear combinations of mean and variance, $m_e + \lambda v_e$, for an arbitrary non-negative λ. We call the solution for this edge cost the λ-optimal solution. This edge cost follows the optimal substructure property and has the property described in Lemma 1, which was also stated in [32].

Lemma 1. *An optimal path occurs among the extreme points of the convex hull for all the O to D paths in the $m - v$ plane if there exists a path that has a mean travel time smaller than the deadline.*

Proof. Let point P on the $m - v$ plane represent the optimal path. Then, there is no path point that has the φ value larger than that of P. Therefore, every other path point must be inside the parabola. Since the parabola is convex, P must be an extreme point. □

With Lemma 1 we can find the optimal solution from λ-optimal solution for a given λ. Since λ-optimal cost satisfies the optimal substructure property, any deterministic shortest path algorithm (e.g. Dijkstra's algorithm or A^* search) will find λ-optimal paths.

6.2.2 Exhaustive Enumeration

In [32], a method for stochastic motion planning was proposed that exhaustively enumerates all the extreme points of the path convex hull. A brief description of the algorithm is as follows: First, find the λ-optimal paths for $\lambda = 0$ and $\lambda = \infty$. If they are the same, it must be the optimal solution. Otherwise, find the λ-optimal path using $\lambda = -\frac{m_0 - m_\infty}{v_0 - v_\infty}$ since this λ value will cause the algorithm to search the entire region completely unless it finds a new path, as illustrated in Figure 6-2 (Left). If no new path is found, the algorithm terminates with the optimal solution being the one with the largest φ value. Otherwise, the newly found path divides the search region into two parts. Then, the λ-optimal search

[1]Linear optimization finds a path that first intersects a straight line when the line is moved in a direction determined by cost parameters

is executed for each region using λ values determined to search each region completely. In this approach, when the number of extreme points is N_e, there will be N_e searches to guarantee that all the extreme points are enumerated. In addition, $N_e - 1$ more searches are needed to conclude that no other paths exist between the extreme points. Thus, the total number of enumerations could be large. Next, we show how to reduce the number of required λ-optimal searches.

6.2.3 Examining Probe Points

If we know that a certain search region's best possible outcome is worse than the current optimal solution, we do not need to execute the costly λ-optimal search for that region. In this section we formalize this point.

Definition 1. *Let the triangular region where a better path can exist be called a* candidate region. *Let the vertex in the middle of the candidate region be called a* probe point. *Candidate regions are illustrated as white triangles $\triangle L_i M_i R_i$, and probe as red points M_i in Figure 6-2 (Left).*

Theorem 1. *If the φ value of the probe point as defined in (6.2) is smaller than the current optimal value, the candidate region does not contain the optimal path.*

Proof. Suppose that a path lies at the probe point. Then, no other point in the candidate region can be an extreme point. The interior points cannot be an optimal solution since the optimal solution occurs at one of the extreme points by Lemma 1. Suppose that a path does not lie at a probe point. Add an imaginary origin to destination path that lies on the probe point. The addition of an imaginary path will not make any difference for searching for the optimal solution since it is not better than the current optimal path. The same argument shows that interior points cannot be optimal solutions. □

By Theorem 1, we can remove from consideration the candidate region if the region's probe point satisfies the condition in Theorem 1. Figure 6-2 (Right) illustrates a case where the right candidate region $\triangle L_2 M_2 R_2$ was removed without any execution of λ-optimal search since the φ value of the probe point M_2 is smaller than that of current optimal path.

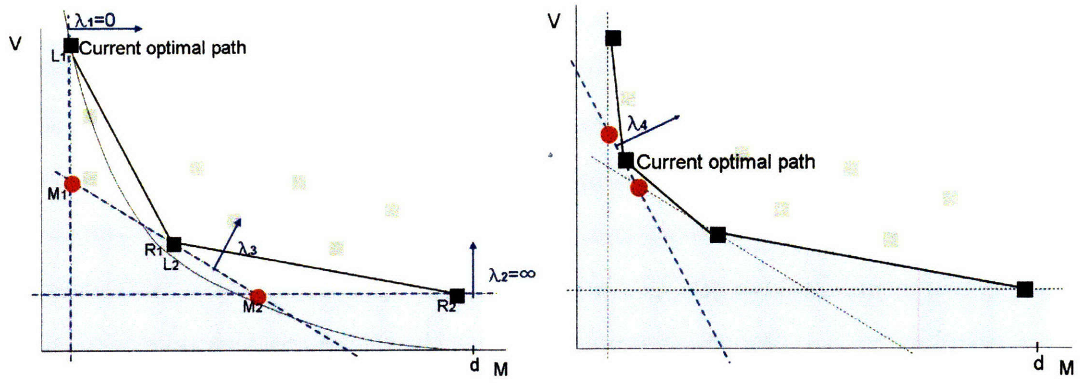


Figure 6-2: (Left) The Result After Three Executions of λ-optimal Searches with $\lambda_1 = 0$, $\lambda_2 = \infty$, and $\lambda_3 = -\frac{m_0 - m_\infty}{v_0 - v_\infty}$. Each black point represents the λ-optimal path for each λ. The gray points represent the paths that are not found yet. The blue regions are guaranteed to contain no path. The white triangles indicate candidate regions for better paths. The red points are the probe points of the regions, which are the intersections of two λ-optimal search lines. (Right) The Result After Another Execution of the λ-optimal Search. A λ-optimal Search was executed only for the left candidate region. The newly found path turns out to be the new current optimal path and the two red points are the new probe points.

The left candidate region $\triangle L_1 M_1 R_1$ was searched since the left probe point gives larger φ value than the current optimal value, and a new path was found as the λ-optimal path. The same procedure is applied to the new candidate regions built by the newly found λ-optimal path until there is no candidate region remaining.

6.2.4 Restricting λ by Upper and Lower Bounds

The λ values that should be searched are limited by upper and lower bounds.

Theorem 2. *The optimal path can be found by searching only with the λ values upper bounded by λ_u, the negative inverse of the tangent to the parabola at the intersection of the 0-optimal search line and the ∞-optimal search line.*

Proof. If the λ-optimal solution is the same as the λ_u-optimal solution for all $\lambda > \lambda_u$, we can trivially find the same path with λ_u instead of $\lambda > \lambda_u$. Suppose that there exists a certain $\lambda > \lambda_u$ for which λ-optimal path (m_λ, v_λ) is different from the λ_u-optimal path $(m_{\lambda_u}, v_{\lambda_u})$. Then, we can say that $m_{\lambda_u} \neq m_\lambda$ and $v_{\lambda_u} \neq v_\lambda$ since $0 < \lambda < \infty$. From the definition of λ-optimal path, $m_{\lambda_u} + \lambda_u v_{\lambda_u} < m_\lambda + \lambda_u v_\lambda$ and $m_{\lambda_u} + \lambda v_{\lambda_u} > m_\lambda + \lambda v_\lambda$.

Rewriting these, we get

$$\lambda_u(v_{\lambda_u} - v_\lambda) < m_\lambda - m_{\lambda_u}, \tag{6.4}$$
$$\lambda(v_{\lambda_u} - v_\lambda) > m_\lambda - m_{\lambda_u}.$$

From the two inequalities we get $(\lambda - \lambda_u)(v_{\lambda_u} - v_\lambda) > 0$ and since $\lambda > \lambda_u$, it follows that $v_{\lambda_u} > v_\lambda$and $m_\lambda > m_{\lambda_u}$.We get an expression for λ_u by taking the derivative of (6.3) and its negative inverse

$$\lambda_u = -1/\frac{\partial v}{\partial m}|_{m=m_0} = \frac{(d-m_0)^2}{2v_\infty(d-m_0)} = \frac{d-m_0}{2v_\infty}. \tag{6.5}$$

From (6.4) and (6.5), and since $d - m_0 > d - m_\lambda$ and $v_\lambda > v_\infty$,

$$\frac{m_\lambda - m_{\lambda_u}}{v_{\lambda_u} - v_\lambda} > \frac{d-m_0}{2v_\infty} > \frac{1}{2}\frac{d-m_\lambda}{v_\lambda}. \tag{6.6}$$

Since $m_\lambda > m_{\lambda_u}$ and $d - m_\lambda > 0$,

$$\frac{1}{\frac{m_\lambda - m_{\lambda_u}}{d-m_\lambda} + 2} < \frac{1}{2} \tag{6.7}$$

From (6.6) and (6.7),

$$\frac{1}{\frac{m_\lambda - m_{\lambda_u}}{d-m_\lambda} + 2} < \frac{m_\lambda - m_{\lambda_u}}{v_{\lambda_u} - v_\lambda}\frac{v_\lambda}{d-m_\lambda}$$

We can rewrite this as follows:

$$\begin{aligned}
\frac{v_{\lambda_u} - v_\lambda}{v_\lambda} &< \frac{m_\lambda - m_{\lambda_u}}{d-m_\lambda}(\frac{m_\lambda - m_{\lambda_u}}{d-m_\lambda} + 2), \\
\frac{v_{\lambda_u}}{v_\lambda} &< (\frac{m_\lambda - m_{\lambda_u}}{d-m_\lambda} + 1)^2, \\
\frac{v_{\lambda_u}}{v_\lambda} &< (\frac{d - m_{\lambda_u}}{d-m_\lambda})^2, \\
\frac{d-m_\lambda}{\sqrt{v_\lambda}} &< \frac{d-m_{\lambda_u}}{\sqrt{v_{\lambda_u}}}.
\end{aligned}$$

Thus, for any $\lambda > \lambda_u$, the λ-optimal solution is worse than λ_u-optimal solution. Thus, there is no need to search the area with the λ that is larger than λ_u. Therefore, whether λ-optimal solution is the same with the λ_u-optimal solution or not we can find the optimal solution by searching with $\lambda <= \lambda_u$. $\square$

Theorem 3. *The optimal path can be found by searching only with the λ values lower bounded by λ_l, the negative inverse of the tangent to the current λ-optimal parabola at the intersection of the current λ-optimal parabola and* 0*-optimal search line.*

Proof. Following the argument as in Theorem 2, suppose that there exists a certain $\lambda < \lambda_l$ for which the λ-optimal path $(m_{\lambda_l}, v_{\lambda_l})$ is different from the λ_l-optimal path (m_λ, v_λ). Then, we can say that $m_{\lambda_l} \neq m_\lambda$ and $v_{\lambda_l} \neq v_\lambda$ since $0 < \lambda < \infty$. From the definition of λ-optimal path, $m_{\lambda_l} + \lambda_l v_{\lambda_l} < m_\lambda + \lambda_l v_\lambda$ and $m_{\lambda_l} + \lambda v_{\lambda_l} > m_\lambda + \lambda v_\lambda$. We can rewrite this as

$$\lambda_l(v_{\lambda_l} - v_\lambda) < m_\lambda - m_{\lambda_l}, \tag{6.8}$$

$$\lambda(v_{\lambda_l} - v_\lambda) > m_\lambda - m_{\lambda_l}. \tag{6.9}$$

Inequalities (6.8) and (6.9) give $(\lambda - \lambda_l)(v_{\lambda_l} - v_\lambda) > 0$ and since $\lambda < \lambda_l$, it follows that $v_{\lambda_l} < v_\lambda$ and $m_\lambda < m_{\lambda_l}$. We get an expression for λ_l by taking the derivative of (6.3) and its negative inverse

$$\lambda_l = -1/\frac{\partial v}{\partial m}|_{m=m_0} = \frac{(d - m_{opt})^2}{2v_{opt}(d - m_0)} \tag{6.10}$$

From (6.8) and (6.10),

$$\frac{m_\lambda - m_{\lambda_l}}{v_{\lambda_l} - v_\lambda} < \frac{\varphi_d(\pi_{opt})^2}{2(d - m_0)}. \tag{6.11}$$

If $\varphi_d(\pi_{opt}) > \varphi_d(\pi_\lambda)$, searching with λ does not give better result then current optimal path. If $\varphi_d(\pi_{opt}) \leq \varphi_d(\pi_\lambda)$,

$$\frac{\varphi_d(\pi_{opt})^2}{2(d - m_0)} \leq \frac{\varphi_d(\pi_\lambda)^2}{2(d - m_0)} < \frac{\varphi_d(\pi_\lambda)^2}{2(d - m_\lambda)} = \frac{d - m_\lambda}{2v_\lambda} \tag{6.12}$$

since $d - m_0 > d - m_\lambda$. From (6.11) and (6.12)

$$\frac{m_\lambda - m_{\lambda_l}}{v_{\lambda_l} - v_\lambda} < \frac{1}{2}\frac{d - m_\lambda}{v_\lambda} \tag{6.13}$$

Since $m_\lambda < m_{\lambda_l}$ and $d - m_\lambda > 0$,

$$\frac{1}{\frac{m_\lambda - m_{\lambda_l}}{d - m_\lambda} + 2} > \frac{1}{2} \tag{6.14}$$

From equation (6.13) and equation (6.14),

$$\frac{1}{\frac{m_\lambda - m_{\lambda_l}}{d - m_\lambda} + 2} > \frac{m_\lambda - m_{\lambda_l}}{v_{\lambda_l} - v_\lambda}\frac{v_\lambda}{d - m_\lambda}$$

Following a similar process as in the proof of Theorem 2, we get

$$\frac{d - m_\lambda}{\sqrt{v_\lambda}} < \frac{d - m_{\lambda_l}}{\sqrt{v_{\lambda_l}}}$$

Thus, for any $\lambda < \lambda_l$, the λ-optimal solution is worse than λ_l-optimal solution. Thus, there is no need to search the area with the λ that is smaller than λ_l. □

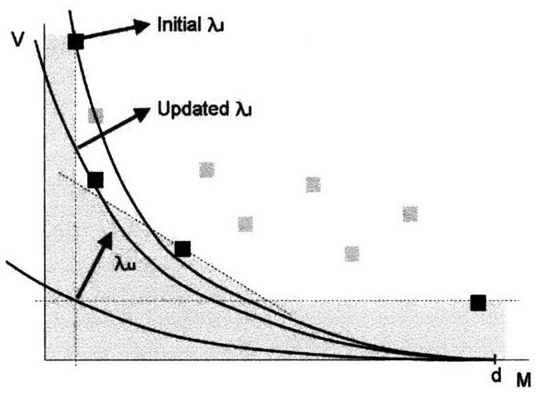


Figure 6-3: Illustration of λ_l and λ_u. λ_l is increased as better λ-optimal path is found.

λ_l and λ_u are illustrated in Figure 6-3. Theorem 1, 2, and 3 lead to the Parametric Search algorithm (see Algorithm 6) for finding the best route that maximizes the probability of arriving at the destination in a given deadline. In line 3 and 4, the 0-optimal and ∞-optimal paths are searched with a shortest path algorithm (e.g. Dijkstra's algorithm or A^*

```
Algorithm 6: PARAMETRIC-SEARCH
Data: Graph with mean and variance of each edge, origin, and destination
Result: The optimal path
1  $bestPath \leftarrow \varnothing$
2  $Rs$ = [] : FIFO queue containing candidate regions.
3  $path_0 \leftarrow$ SEARCH-$\lambda$-OPTIMAL-PATH(0)
4  $path_\infty \leftarrow$ SEARCH-$\lambda$-OPTIMAL-PATH($\infty$)
5  if $path_0 == path_\infty$ then
6      return $path_0$
7  $Rs.push(Region(l : path_0, r : path_\infty, p : (path_0.mean, path_\infty.var)))$
8  calculate $\lambda_l$ and $\lambda_u$
9  while $(R \leftarrow Rs.pop()) \neq \varnothing$ do
10     if $R.probe.\varphi < bestPath.\varphi$ then continue
11     $\lambda \leftarrow -\frac{R.l.mean - R.r.mean}{l.var - r.var}$
12     if $\lambda \geq \lambda_u$ then
13         if $\lambda_u$ was not searched then $\lambda \leftarrow \lambda_u$
14         else continue
15     if $\lambda \leq \lambda_l$ then
16         if $\lambda_l$ was not searched then $\lambda \leftarrow \lambda_l$
17         else continue
18     $path \leftarrow$SEARCH-$\lambda$-OPTIMAL-PATH($\lambda$)
19     if $path \neq R.l$ and $path \neq R.r$ then
20         if $path.\varphi > bestPath.\varphi$ then
21             $bestPath \leftarrow path$, update $\lambda_l$
22         calculate $probe_l$ and $probe_r$
23         if $probe_l.\varphi > bestPath.\varphi$ then
24             $Rs.push(Region(l : R.l, r : path, p : probe_l))$
25         if $probe_r.\varphi > bestPath.\varphi$ then
26             $Rs.push(Region(l : path, r : R.r, p : probe_r))$
27 return $bestPath$
```

search). If the two found paths are the same the algorithm terminates. If they are different, the first candidate region consisting of the three points denoted in line 7 is pushed onto the queue. Candidate regions are evaluated for searching. The conditions in line 10, 23, and 25 come from Theorem 1, and those in line 12 and 15 from Theorem 2 and 3, respectively. If the candidate region does not need to be searched, the algorithm continues with the next region. Otherwise, the region is searched with the λ value determined by the left and right path of the region (line 11) and possibly modified by the upper and lower bounds (line 13 and 16) in line 18.

6.2.5 Correctness

Algorithm 6 finds the optimal solution in a finite number of λ-optimal searches. The paths in the region we exclude from the exhaustive enumeration using the extreme points cannot be optimal by Theorem 1. The paths in the region we excluded using the upper and lower bound of λ cannot be optimal due to Theorems 2 and 3. Since the number of required λ-optimal searches is upper bounded by $2N_e - 1$ as described in Section 6.2.2, the algorithm finds the optimal solution in a finite number of searches.

6.2.6 Running Time

[32] showed that the running time of the exhaustive λ-optimal search is $n^{\Theta(log\ n)}$, where the number of nodes is n (since there are $n^{\Theta(logn)}$ extreme points in the worst case ([12]) and we can find one extreme point in $O(n^2)$ using any standard shortest path algorithm.) We have observed empirically a speedup factor between 10 and 20. The running time is loosely upper bounded by $n^{\Theta(log\ n)}$. We are currently working on a theoretical characterization of the speedup.

6.3 A Pruning Algorithm

6.3.1 Motivation

Since the cost function for stochastic motion planning is not linear, the optimal substructure property does not hold. However, in [29], the authors provided some pruning methods. In this section we give a condition that can be used to rank any two subpaths, which is specific to the case where the edge cost follows a Gaussian distribution and the goal is maximizing the probability of reaching the destination in a given deadline.

Definition 2. *Suppose paths π_1 and π_2 reach intermediate node N. π_1* is superior to *π_2 iff for any trajectory t from N to D, π_1 followed by t is better than π_2 followed by t.*

The following theorem describes the subsuperiority condition.

Theorem 4. *Let π_1 and π_2 be two different O-to-N path with mean and variance (m_1, v_1) and (m_2, v_2), respectively. Then π_1 is superior to π_2 if $m_1 < m_2$ and $\frac{d-m_1}{\sqrt{v_1}} > \frac{d-m_2}{\sqrt{v_2}}$.*

Proof. Let the mean and variance of an arbitrary N-to-D path be m_{ND} and v_{ND} respectively.

Case 1) when $v_1 \leq v_2$:

From the inequality relationships between m_1, m_2 and v_1, v_2,

$$\frac{d-(m_1+m_{ND})}{d-(m_2+m_{ND})} > 1 \text{ and } \frac{\sqrt{v_1+v_{ND}}}{\sqrt{v_2+v_{ND}}} \leq 1.$$

Thus, it follows that

$$\frac{d-(m_1+m_{ND})}{\sqrt{v_1+v_{ND}}} > \frac{d-(m_2+m_{ND})}{\sqrt{v_2+v_{ND}}}.$$

Since this inequality holds for any N-to-D path, π_1 is superior to π_2.

Case 2) when $v_1 > v_2$:

From the inequality relationship between m_1, m_2 and v_1, v_2, it follows that

$$\frac{d-(m_1+m_{ND})}{d-(m_2+m_{ND})} > \frac{d-m_1}{d-m_2} \text{ and } \frac{\sqrt{v_1}}{\sqrt{v_2}} > \frac{\sqrt{v_1+v_{ND}}}{\sqrt{v_2+v_{ND}}}.$$

Since we have

$$\frac{d-m_1}{d-m_2} > \frac{\sqrt{v_1}}{\sqrt{v_2}},$$

it follows that

$$\frac{d-(m_1+m_{ND})}{d-(m_2+m_{ND})} > \frac{\sqrt{v_1+v_{ND}}}{\sqrt{v_2+v_{ND}}}.$$

Thus,

$$\frac{d-(m_1+m_{ND})}{\sqrt{v_1+v_{ND}}} > \frac{d-(m_2+m_{ND})}{\sqrt{v_2+v_{ND}}}.$$

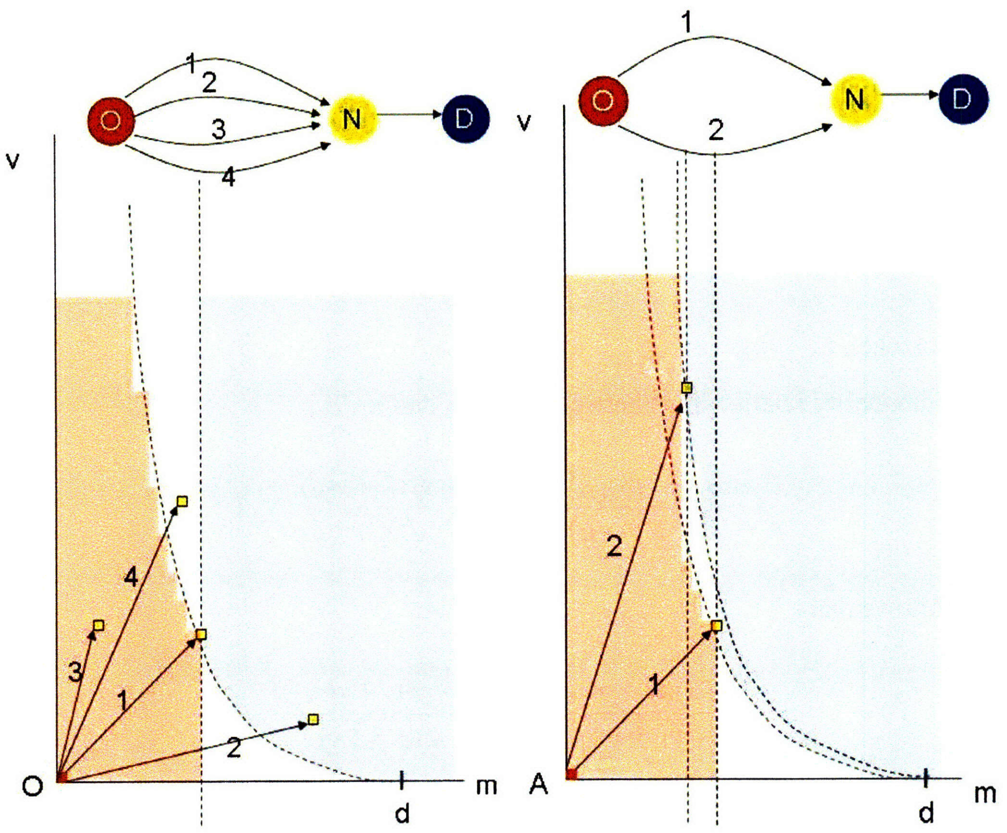


Figure 6-4: (Left) Illustration of Subsuperiority Conditions. Let O be an origin and D a destination. Given a path 1 for a certain node N, any O to N path lying in the blue area (path 2) is inferior to path 1. The path 1 is inferior to the other path (path 3) when it lies in the red area. If another path (path 4) lies in the white area, both paths coexist. (Right) Illustration of a Coexistance Area for Three Paths. When path 1 and path 2 coexist, the region for another path to coexist is the white area, which is reduced as the number of coexisting points increases.

Since this inequality holds for any N-to-D path, π_1 is superior to π_2. □

Figure 6-4 (Left) illustrates the subsuperiority relationships between subpaths. The paths lying in the red region are superior to path 1 and those lying in the blue region are inferior to path 1. The paths that lie in white area coexist with path 1. Coexisting paths are not desirable since each extension of every coexisting path should be examined. However, the probability that many paths coexist is not high since the area allowing coexistance shrinks as more paths coexist, as shown Figure 6-4 (Right).

Theorem 4 leads to the algorithm shown in Algorithm 7.

Algorithm 7: PRUNING-ALGORITHM

```
Data: Graph with mean and variance of each edge, origin $O$, destination $D$
Result: The optimal path
1  $paths$[i]: An array of subpath lists reaching node $i$
2  $openPaths$ = []: FIFO queue for open subpaths
3  $openPaths.push(O.nodeID)$
4  while ($curNodeID \leftarrow openPaths.pop()$) $\neq \varnothing$ do
5      $path \leftarrow paths[curNodeID]$
6      while $path \neq \varnothing$ do
7          if $path.closed$ then $continue$
8          $neighbors \leftarrow getNeighbors(path)$
9          foreach $n$ in $neighbors$ do
10             $inferior \leftarrow False$
11             $stored \leftarrow paths[n.nodeID]$
12             while $stored \neq \varnothing$ do
13                 if $n$ is superior to $stored$ then
14                     $paths[n.nodeID].remove(stored)$
15                 else if $n$ is inferior to $stored$ then
16                     $inferior \leftarrow True$
17                 $stored \leftarrow stored.next$
18             if not $inferior$ then
19                 $paths[n.nodeID].insert(n)$
20                 $openPaths.push(n.nodeID)$
21         $path.closed \leftarrow True$
22         $path \leftarrow path.next$
23 return the best path in $paths[D.nodeID]$
```

For every open paths (line 4), all the extended paths to the next nodes (line 8) are compared with the stored paths. If the new extension is superior to a stored path, the stored path is removed (line 13). If the new extension is not inferior to any stored paths, it is inserted (line 18).

6.3.2 Correctness

Any path with a loop is inferior to a path that does not have a loop since a loop adds positive mean and variance to the loopless path. Note that path with larger mean and larger variance is inferior. The algorithm stops in a finite number of steps since paths with loops are pruned. Since the extension of pruned subpath cannot be better than that of superior subpath, the pruned path cannot be optimal. Since every non-pruned path was compared for optimality, Algorithm 7 finds the optimal path in a finite number of steps.

6.4 Algorithm Evaluation

We have evaluated empirically the performance of Algorithms 6 and 7 against the exhaustive λ-optimal search proposed in [32] using simulation data and real data from the taxi database.

6.4.1 Experimental Data

Grid Structure

The simulation data set is a square bidirectional grid structure where each edge has a random mean and variance uniformly distributed between 0 and 1. Grid structures of size from 10×10 to 100×100 were used. The origin and destination are two diagonally opposite points.

Physical Road Network

The Delay Statistics Map built using CarTel the taxi database was used as a physical test bed. The map has about 29000 nodes and 39000 edges. It is dense around the Greater Boston area and granular in rural area of Massachusetts.

6.4.2 Running Time

The running time of Algorithms 6 and 7 were compared with the exhaustive λ-optimal searches using the two data sets above. Two variations of Algorithm 1, where only one of the two methods introduced in Section 6.2.3 and 6.2.4 were used to examine the effect of each method. Figure 6-5 shows the results on the simulation data. Algorithm 6 with both methods runs fastest. The speedup is by at least a factor of 10 over the algorithm in [32].

The speedup is due to reduced amount of λ-optimal searches. For the algorithm in [32], the number of λ-optimal searches gradually increased from 17 to 119 as the number of nodes increase from 100 to 10000, whereas it increased from 5 to 7 for Algorithm 6. Algorithm 7 runs faster than the algorithm in [32], but not as fast as Algorithm 6.

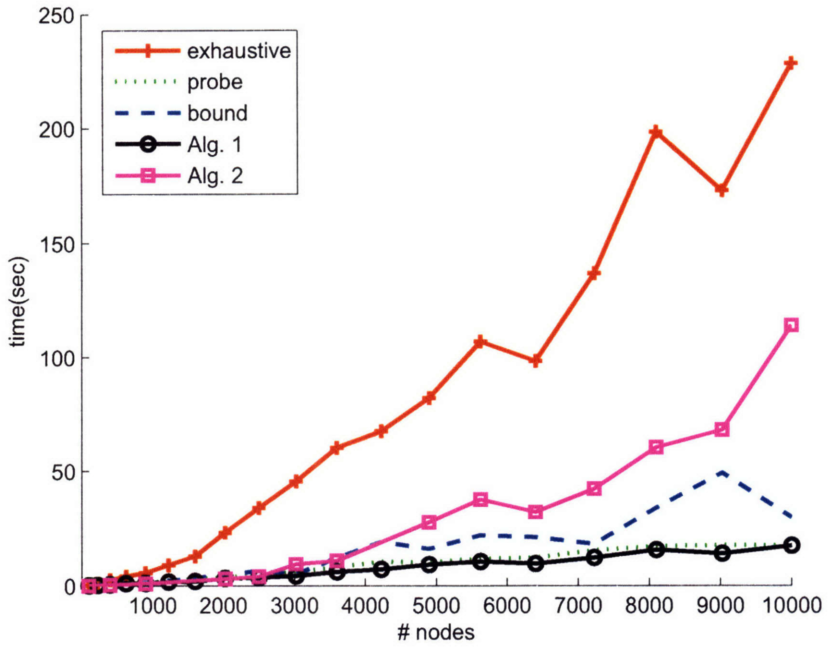


Figure 6-5: Running Time Measured at the Square Bidirectional Grid Structure, where each edge has a random mean and variance between 0 and 1 with a deadline of half grid size. "exhaustive" is the exhaustive λ-optimal search, "probe" is just applying the candidate region probing method, "bound" is just applying the bounds of λ, and "Alg. 1" is Algorithm 6. "Alg. 2" is Algorithm 7.

The running time of Algorithm 6 on the Delay Statistics Map for a route 144km long is 14 seconds with 5 λ-optimal searches when the deadline is 3 hours. The same query took 178 seconds when we used the exhaustive λ-optimal search yielding 75 λ-optimal searches.

Chapter 7

Using Current Information

In this chapter we give a method using current information for the road segments as it is observed. For now we do not expect to observe enough current information for all the roads, which is required to plan the optimal path since real-time coverage by 28 taxis is limited. Thus, we do not use complicated estimation and prediction methods but use current information as supplementary information. The method suggested in this chapter uses the observed current information and propagates the information as far as the road segments are statistically related. We will develop a more sophisticated algorithm that uses both historical and current information, which can be used when we have a significant amount of current information.

7.1 Current Information Incorporation

In this section we explain how we utilize the current information for path planning by incorporating the current information into the historical information. We update the database by the information observed in real time. When the stochastic shortest path algorithm runs, we substitute the historical information with this current information for the road segments that have updated information for a predefined time window. That is, we can substitute the mean travel time with the current observed travel time, and the variance of the travel time with a smaller value than the historical variance. Since the current observation will be related to other connected road segments, we use this method for other road segments where the current information was propagated from adjacent road segments by the algorithm introduced in the next section.

7.2 Current Information Propagation

In this section we provide an algorithm that propagates the current information to statistically related road segments. From the taxi traces, we mark the transition from one road group to another, and keep the number of incidences from each adjacent road as shown by an example in Figure 7-1. Then, we propagate a velocity measurement on one road group to others if the statistics are similar and the amount of flow is mostly preserved.

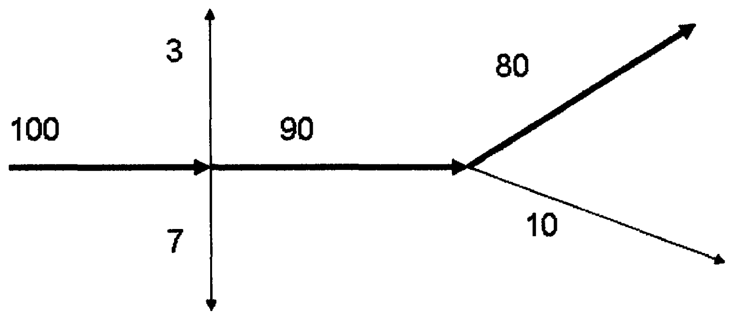


Figure 7-1: Example of Flow Preservation. Since the flow is preserved through the thick arrows, we can consider propagating current information observed at one of those thick arrows to other thick arrows.

Algorithm 8 shows how the current observed velocity is propagated through related groups. Algorithms 9 - 12 are the functional blocks called in Algorithm 8. When GPS probe vehicles update current information for the road groups where the vehicles are, the information is propagated by Algorithm 8. Algorithm 8 recursively calls itself (line 8) if the propagation conditions are satisfied (line 6) for each group connected to the current group in either forward or backward directions (line 4). The propagation stops if the maximum amount of propagation defined by the first call of the Algorithm 8 is reached (line 2).

Algorithm 9 returns the connected groups to the current group. If *direction* is forward, the groups that are forward with regard to the vehicle's moving direction are returned.

Algorithm 10 examines whether the statistics of the adjacent two groups are similar. The algorithm compares both the average velocity of each group and the travel time fluctu-

```
Algorithm 8: PROPAGATE
  Data: $direction$ : indicator of the forward or backward propagation,
  $velocity$ : the observed velocity,
  $updateTime$ : the time of the observation,
  $group$ : the group where the velocity observed,
  $table$ : the Groups Table corresponding to the time of day,
  $count$ : the propagation count
  Result: Updated current information
1 $count \leftarrow count - 1$
2 if $count < 0$ then
3     return
4 $newGroups \leftarrow$ FIND-GROUPS($direction, group, table$)
5 foreach $newGroup \in newGroups$ do
6     if IS-SIMILAR($group, newGroup$) and
      IS-FLOW-CORRELATED($direction, group, newGroup$) then
7         UPDATE-CURRENT-INFO($velocity, updateTime, newGroup, table$)
8         PROPAGATE($direction, velocity, updateTime, group, table, count$)
```

```
Algorithm 9: FIND-GROUPS
  Data: $direction, group, table$
  Result: a list of groups
1 return the list of groups that are connected to the group depending on the direction
```

ation from the mean for each group.

Algorithm 11 checks if the amount of cars (flow) is preserved from one group to another. If the flow is preserved between two groups, there is high chance that the statistics of the two groups are similar.

Algorithm 12 updates the current information fields of the Groups table corresponding to the current time of day in the database.

Algorithm 10: IS-SIMILAR

```
Data: group, newGroup
Result: True or False
1 if group.distance/group.mean is similar to
  newGroup.distance/newGroup.mean and group.std/group.mean is similar to
  newGroup.std/newGroup.mean then
2     return True
3 else
4     return False
```

Algorithm 11: ISFLOWCORRELATED

```
Data: direction, group, newGroup
Result: True or False
1  if direction == forward then
2      if the flow through newGroup followed by group is more than 30 % of the total
       flow through group then
3          return True
4      else
5          return False
6  else
7      if the flow through group followed by newGroup is more than 30 % of the total
       flow through newGroup then
8          return True
9      else
10         return False
```

Algorithm 12: UPDATE-CURRENT-INFO

```
Data: velocity, updateTime, newGroup, table
Result: Updated Groups Table table
1 update the meanTime field of newGroup in table with the time calculated from
  velocity, and the updateTime field with the given updateTime.
```

Figure 7-2 shows an example of current information propagation by the algorithm discussed in this chapter. The red lines in Figure 7-2 indicate the road segments where current information is observed and orange lines denote the road segments where observed information was propagated.

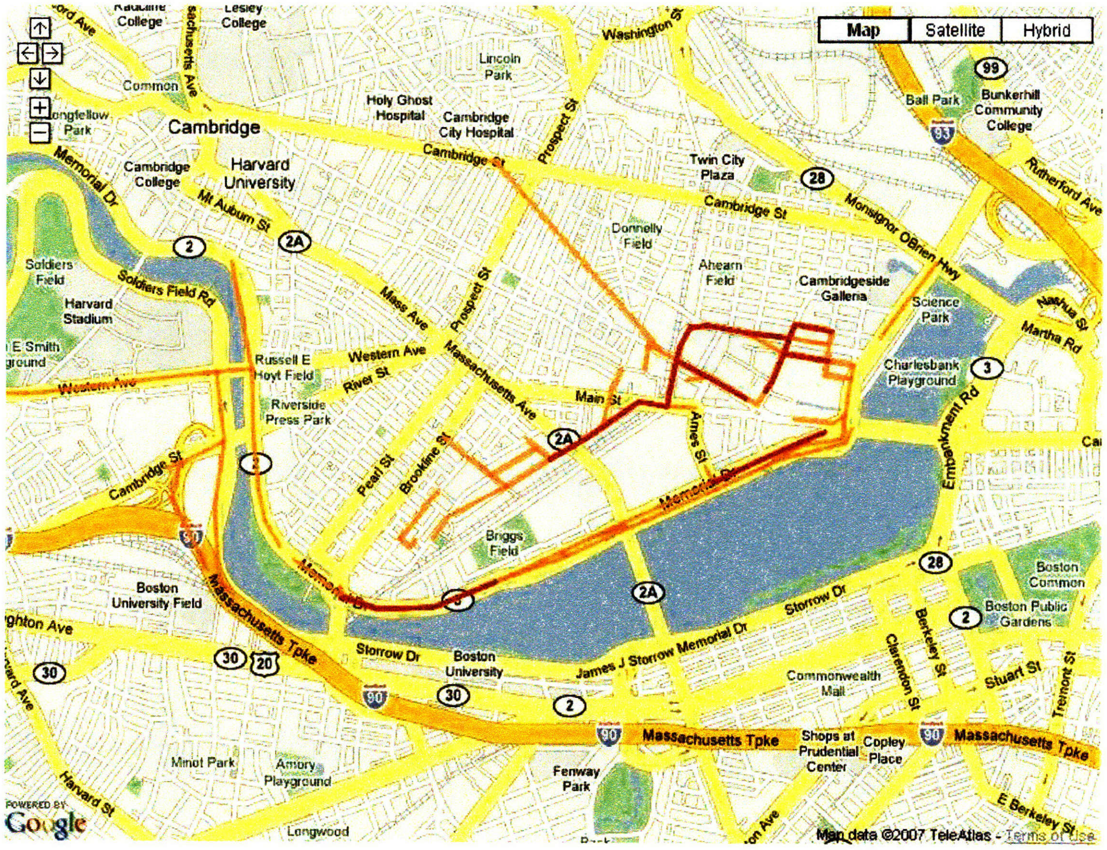


Figure 7-2: An Example of Current Information Propagation

Chapter 8

The Intelligent Traffic System

8.1 System Overview

The traffic information system consists of a database called *RoadNetDB* and a Web server. The Web server is implemented with Common Gateway Interface (CGI) consisting of a frontend Graphical User Interface (GUI) module and a backend processing module, which accesses RoadNetDB for information queries. Server side programs were written in python and client side programs were written in javascript. The client side program uses the Google map open API [3] for visualizing the paths. For a faster route search, a database was organized using spatial indexes, highly accessed tables were cached from the database, the highly used parts were implemented with faster language (C programming language), and user requests were processed asynchronously for better user experience.

8.2 Database

We established a database system on a Linux computer with 2.4 GHz CPU and 2 GB memory. The database system we used is MySQL 5.0.32 database, which provides spatial functionalities so that spatial queries are optimized with an R-tree structure. The main tables in RoadNetDB are *Segments Table* corresponding to the Geographic Map and *Groups Table* corresponding to the Delay Statistics Map. Each row in these tables represents each edge in those Maps. We call the edges in the Geographic Map simply *segments* and those in the Delay Statistics Map as *groups*. First, we implemented a java program that extracts the required information from the Navteq or TIGER/Line data files . We first started the project with the TIGER/Line files, but because of the lack of overhead crossing, one-way, and restricted maneuvers information, etc., we switched to the Navteq sample data for northeastern regions of the United States. The program extracts data fields for each road segment, including the coordinate of the two end points, street name, street category, one-way information, restricted maneuvers, etc., and stores those into data files. Second, the extracted data files are uploaded to RoadNetDB, generating the Segments Table representing the Graphical Map as shown in Table 8.1. As a result, there are a total of 602268 segments in the Segments Table.

Table 8.1: Segments Table: represents each road segment in the Geographic Map

Field	Description
SegmentID	unique id of each segment
StartPoint	latitude, longitude, z level of the start point
EndPoint	latitude, longitude, z level of the end point
StreetName	name of the street
Distance	the distance calculated by the two end point
Category	a road category
Line	a MySQL LineString spatial data format value for the end-to-end line
GroupID	unique id of group which this segment belongs to
GroupDirection	the relative direction of this segment compared to direction of group

Next, we conduct the following sequel of processes to generate the Delay Statistics Map. First, we run a map matching python script that matches every taxi trajectory to the segments as defined in the Segments Table according to Algorithm 1. Each row of the Map Matching Table is a segment matched from trajectories. Table 8.2 shows the fields in the Map Matching Table. During this map matching process, the time taken to travel the segments are kept. The travel time for each segment can be found by the difference from subsequent "time" field. The "StartFlag" in Table 8.2 indicates whether consecutive map-matching results are in a continuous travel or not. It is "1" when a new travel starts, "2" when the last segment of the previous map-matching is the same with the first segment of the current map matching, and "3" when the last segment of the previous map matching is connected to the first segment of the current map matching. Second, a python script for aggregation runs to generate the Groups Table according to Algorithm 4. The fields in Groups Table are shown in Table 8.3. Finally, a python script for assigning taxi delay data into the groups is run to assign the statistics for each group according to Algorithm 5. Groups Tables (Table 8.3) contain mean and variance of the travel time for each group calculated from the taxi GPS data per hour and per day.

Table 8.2: Map Matching Table: describes a road sequence of segments driven by cars

Field	Description
SegmentID	the matched segment id
Direction	the direction of the trip relative to the direction of the segment
Time	the time when the car enters the segment
Velocity	velocity' is the average velocity for 2 minute trajectory
StartFlag	indicates the connection between subsequent map-matching
Car	the label of the car

Table 8.3: Groups Table: represents the delay statistics for each group in the Delay Statistics Road Map

Field	Description
GroupID	unique id of each group
StartPoint	latitude, longitude, z level of the start point
EndPoint	latitude, longitude, z level of the end point
StreetName	name of the street
Distance	the distance calculated by the two end point
Category	a road category
Line	line object of MySQL spatial feature
MeanTimeNormal	mean of the normal direction
VarTimeNormal	variance of the normal direction
NumSamplesNormal	number of samples of the normal direction
MeanTimeOpposite	mean of the opposite direction
VarTimeOpposite	variance of the opposite direction
NumSamplesOpposite	number of samples of the opposite direction

8.3 Backend Processing Unit

8.3.1 Optimal Path Query

In this section the backend query processing for optimal route search is explained. As is explained before, we keep the delay statistics (mean and variance) not for every road segment defined by the Geographic Map but for the road groups defined by the Delay Statistics Map. Thus, delay sensitive routing is basically done through the Delay Statistics Map. Since the origin and destination are not necessarily among the nodes of the Delay Statistics Map, it is required to connect the origin and destination node to the nearest segment group in the Delay Statistics Map. Thus, the path search is done in three steps: First, we find a shortest path from the start point to the point in the Delay Statistics Map by running A^* search or Dijkstra's algorithm on the Geographic Map. Second, a similar process is done for the end point in the reverse direction. Finally, we find a route connecting the origin side group point and the destination side group point through Delay Statistics Map using the algorithms developed in Chapter 6. The data type and functions used to implemented this is as follows.

Data Type

A path is represented by a linked list of nodes, where the Node object represents nodes as follows:

- **Node** : Node has the following fields as member.

  · **x, y, z** : the coordinate of the node. x, y represents latitude and longitude, respectively, and z represents the Z-level to distinguish multiple layer road structures.

  · **currentCost** : the current cost from the start node to the current node.

  · **estimatedCost** : the estimated cost from the current node to the end node, which is used as a heuristic cost of A^* search algorithm.

  · **segmentID, streetName** : the id and street name of the segment that connects the previous node and current node.

  · **mean, variance** : the mean and variance of the travel time for the segment that connects the previous node and current node.

  · **previousNode** : the pointer to the previous node.

Functions

- **getClosestPoint(point)** Returns the closest segment end point for origins and destinations given by user input. This queries the information from the Segments Table in RoadNetDB using the MySQL spatial search functionality for the fast look-up.

- **getShortestPathToDSM(startNode, endNode, direction)** Finds the path from "startNode" to the Delay Statistic Map by A^* search or Dijkstra's algorithm if "direction" is forward. If "direction" is backward, the path from the Delay Statistic Map to the "endNode" is found. "startNode" and "endNode" are the points in the Geographic Map.

- **getOptimalPathOnDSM(startNode, endNode, deadline, λ, mode, useCurrentInfo)** Finds the path from "startNode" to "endNode" using the stochastic shortest path algorithm developed in this thesis. "startNode" and "endNode" are the points in the Delay Statistics Map. "mode" indicates which algorithm to use. "useCurrentInfo" indicates whether current information will be used or not.

- **getNodeNeighbors(table)** Queries the next nodes connected to the current node. "table" is the Geographical Map when it is called from "getShortestPathToDSM", and it is the Delay Statistics Map when it is called from "getOptimalPathOnDSM". Because this method is frequently called for the Delay Statistics Map, we cached the neighbors of each node in a file to reduce the execution time for database queries.

8.3.2 Map Matching Result Query

In this section the backend query processing for finding matched road segments given a sequence of GPS coordinates that represents a trajectory. For each subtrajectory of the given trajectory, first, the nearby segments of each point in the subtrajectory are queried. Second, the best matching segment to the subtrajectory is found among the nearby segments. Finally, the found segment is examined whether it is connected to previous matches. The following functions explain each step of this query process.

Functions

- **getClosestStreets(point, length)** Returns the street segments that lie in the square region defined by a center point "point" and the length of a side "length". This function obtains the road segments satisfying the input condition by querying the Segments Table of the RoadNetDB. For a fast look up, MySQL spatial query commands using "Polygon" and "MBRIntersects" are generated. This SQL query returns only the road segments that intersects with the polygon representing the square area.

- **getStreetCandidate(streetSet, subtrajectory, agingIDs, agingCounts)** Returns the locally best segment for the "subtrajectory" among the nearby road segments in "streetSet" considering the aging factors for the road segments in the "agingIDs" list. This is the implementation of the local matching algorithm, Algorithm 2

- **processCandidate(streetSet, subtrajectory, segments, candidate, agingIDs, agingCounts)** Returns whether locally best candidate "candidate" is connected to the matched segment list "segments". If connectivity does not hold, four more local best candidates are examined until connectivity holds. If the connectivity is satisfied with none of these candidates, the last matched segment in "segments" is removed since it is regarded as a wrong match.

8.3.3 Congestion Area Query

In this section the processing used for retrieving the congestion information is explained.

Functions

- **getSegmentsByConditions(condition, center, length)** Returns the road segments that satisfy the travel time statistics defined by the input conditions "condition". This function generates a MySQL spatial query that retrieves the rows from the Groups Table that correspond to "condition" and lie inside the square area specified by "center" and "length".

8.3.4 User Route Upload and Analysis Query

In this section the processing needed for user file upload and trajectory retrieval and visualization is explained.

Data type

- **GPXTrackPoint** : This represents the GPX track point which is parsed from GPX files. GPXTrackPoint has the following members.

  · **lat, lon, ele** : the latitude, longitude, and elevation of the GPS point.

  · **time** : the time of the GPS point

Functions

- **uploadFile()** Uploads the users' track-log files to the log repository directory. This function call is usually followed by the "parseAndStoreInDB" function call with a filename argument specifying this file.

- **parseAndStoreInDB(filename)** Parses the track-log files specified by "filename" to generate a sequence of GPXTrackPoint objects. The GPS points are inserted into the table named UserTraces in the RoadNetDB, which has *userID, tripID, lat, lon, time* as its columns. The summary of the trip represented by this GPS points is inserted into another table named TripSummaries, which has *userID, title, hour, day, duration, startTime, endTime, startLat, startLon, endLat, endLon,* and *userDefinedLabel.*

- **parseAndShow(filename)** Parses the track-log files specified by "filename" and returns the GPS point string, which is used by the GUI module to visualize the corresponding trajectory color coded by speed on the map.

- **getUserTripSummary(userID)** Returns the summary information of the trip that the user uploaded. This function checks the cookies to confirm that the user is authorized to look up the data for the "userID". This function generates an SQL query to the TripSummaries table in the RoadNetDB.

- **getUserTrace(userID, tripID)** Returns the trajectory corresponding to the "userID" and "tripID". This function checks the cookies to confirm that the user is authorized to look up the data. This function generates an SQL query to the UserTraces table in the RoadNetDB.

8.4 Graphical User Interface

All user requests are processed by Asynchronous Javascript And XML (AJAX) technology to provide better user experience. When a user clicks buttons, the corresponding javascript methods are called, and the user request are processed by server side python programs. The results are notified to the javascript callback functions asynchronously. The Google map open API [3] was used to visualize the results on a map image.

8.4.1 Optimal Path

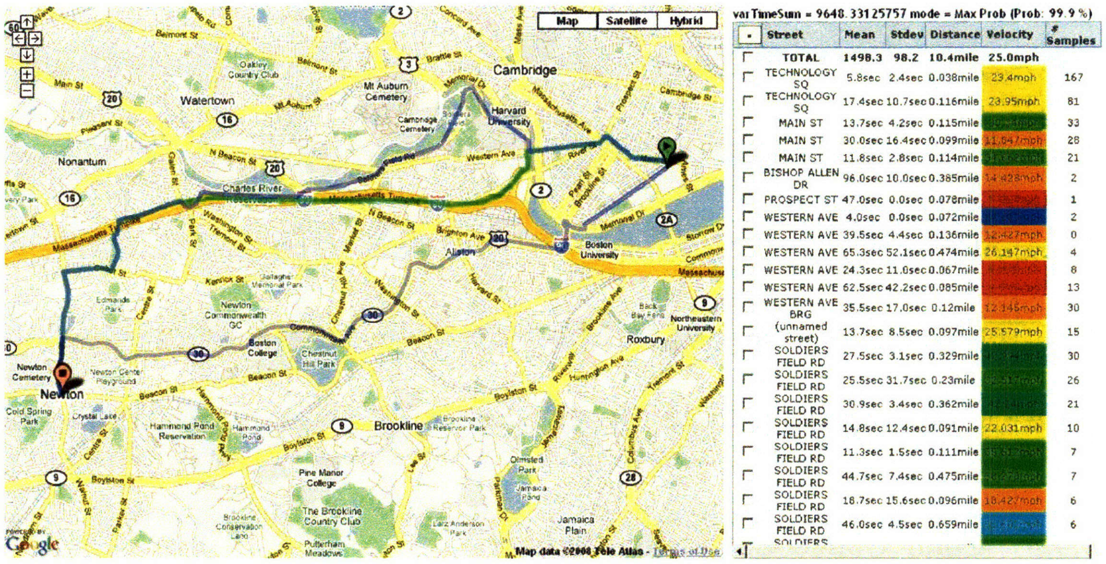
Optimal path finding interface provides an easy travel-condition-input form and an effective display of the result. There are two ways to enter an origin and a destination. One is to enter the address in the "Address" text box in the top row of the input form. Another way to enter an origin and a destination is by clicking the corresponding spot on the map. The first click generates an origin point, and the next click generates a destination point. Users can navigate over the map either by the control panel at the left top side of the map or by drag and drop. Users can select the "Hour of day" and "Day of week" when they want to travel. The "Deadline" is the time in minutes in which they should finish the travel. There are three buttons that find different optimal routes:

Minimum Distance : The route that has the minimum distance;

Minimum Expected Time : The route that would be the fastest route on average;

Maximum Probability : The route that gives the highest chance of reaching the destination in the deadline.

Users can also check what Google Maps provides by clicking the "Google" button on the right top side of the input form. Users can also select "Use current information" to use current information, though it is using sparse current data for now.



varTimeSum = 9648.33125757 mode = Max Prob (Prob: 99.9 %)

Street	Mean	Stdev	Distance	Velocity	# Samples
TOTAL	1498.3	98.2	10.4mile	25.0mph	
TECHNOLOGY SQ	5.8sec	2.4sec	0.038mile	23.4mph	167
TECHNOLOGY SQ	17.4sec	10.7sec	0.116mile	23.95mph	81
MAIN ST	13.7sec	4.2sec	0.115mile	[illegible]	33
MAIN ST	30.0sec	16.4sec	0.099mile	11.647mph	28
MAIN ST	11.8sec	2.8sec	0.114mile	[illegible]	21
BISHOP ALLEN DR	96.0sec	10.0sec	0.385mile	14.428mph	2
PROSPECT ST	47.0sec	0.0sec	0.078mile	[illegible]	1
WESTERN AVE	4.0sec	0.0sec	0.072mile	[illegible]	2
WESTERN AVE	39.5sec	4.4sec	0.136mile	12.427mph	0
WESTERN AVE	65.3sec	52.1sec	0.474mile	26.147mph	4
WESTERN AVE	24.3sec	11.0sec	0.067mile	[illegible]	8
WESTERN AVE	62.5sec	42.2sec	0.085mile	[illegible]	13
WESTERN AVE BRG	35.5sec	17.0sec	0.12mile	12.145mph	30
(unnamed street)	13.7sec	8.5sec	0.097mile	25.579mph	15
SOLDIERS FIELD RD	27.5sec	3.1sec	0.329mile	[illegible]	30
SOLDIERS FIELD RD	25.5sec	31.7sec	0.23mile	[illegible]	26
SOLDIERS FIELD RD	30.9sec	3.4sec	0.362mile	[illegible]	21
SOLDIERS FIELD RD	14.8sec	12.4sec	0.091mile	22.031mph	10
SOLDIERS FIELD RD	11.3sec	1.5sec	0.111mile	[illegible]	7
SOLDIERS FIELD RD	44.7sec	7.4sec	0.475mile	[illegible]	7
SOLDIERS FIELD RD	18.7sec	15.6sec	0.096mile	18.427mph	6
SOLDIERS FIELD RD	46.0sec	4.5sec	0.659mile	[illegible]	6
SOLDIERS					

Figure 8-1: The User Interface to the Traffic Information System with Highlighted Paths and Travel Time Estimates. Optimal Path Query Result for a Travel from MIT ("O") to Newton, MA ("D") at 5~6 pm on Weekdays.

Figure 8-1 shows an example of the paths found by our optimal path planning GUI. The top blue line is the maximum arrival probability path given a deadline of 30 minutes. The green line is the minimum expected time path. The bottom blue line is the path Google Maps give.

8.4.2 Trajectory and Map-Matched Streets

Current and historical trajectories and their corresponding road segments found by our map-matching algorithm can be shown using a map-matching GUI. A trajectory to match onto the map is selected by the user input: the label of the car, the start time, and the duration. Figure 8-2 shows an example of the taxi trajectory query result.

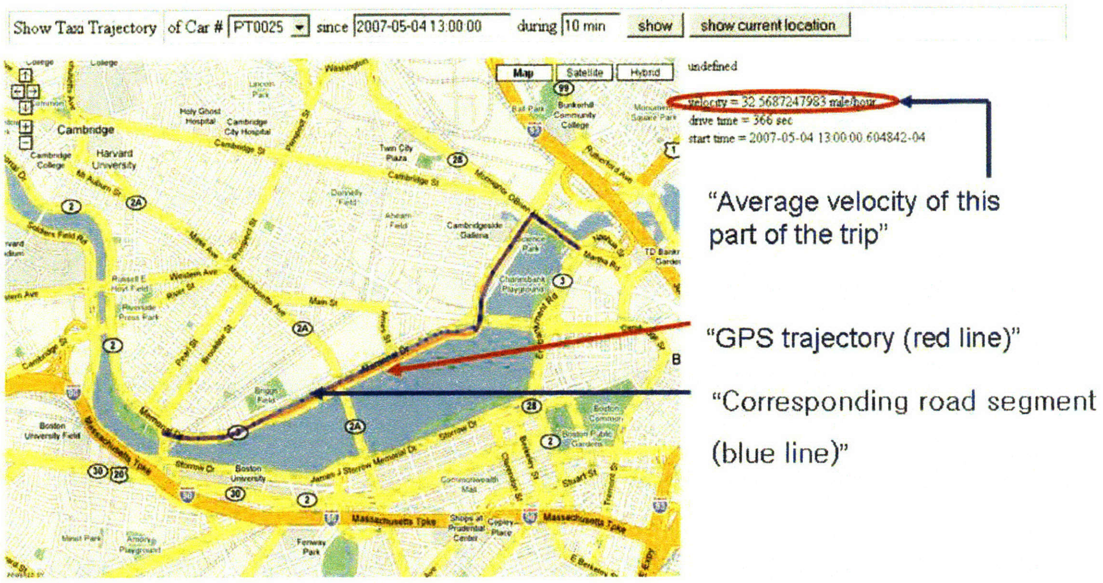

Show Taxi Trajectory of Car # PT0025 since 2007-05-04 13:00:00 during 10 min
show
show current location
Map
Satellite
Hybrid
undefined
velocity = 32.5687247983 mile/hour
drive time = 366 sec
start time = 2007-05-04 13:00:00.604842-04
"Average velocity of this part of the trip"
"GPS trajectory (red line)"
"Corresponding road segment (blue line)"

Figure 8-2: Map Matching Result. The red line is actual GPS points reported from the CarTel node and blue line is a sequence of segments that correspond to the GPS trajectory.

8.4.3 Color Coded Congestion Map

Figure 8-3 visualizes congestion spots. Users can select the level of congestion to query both by the travel time variation (the ratio of standard deviation and mean) and by the mean velocity. Users can also select the area of interest by center point and width. For the top figure the average velocity becomes lower as color changes from blue to green, yellow, orange, red. For the bottom figure the fluctuation of the travel time becomes larger as color changes from blue to green, yellow, orange, red. Thus, red road segments are hot spots, where average velocity is low and the variation of the travel time is large.

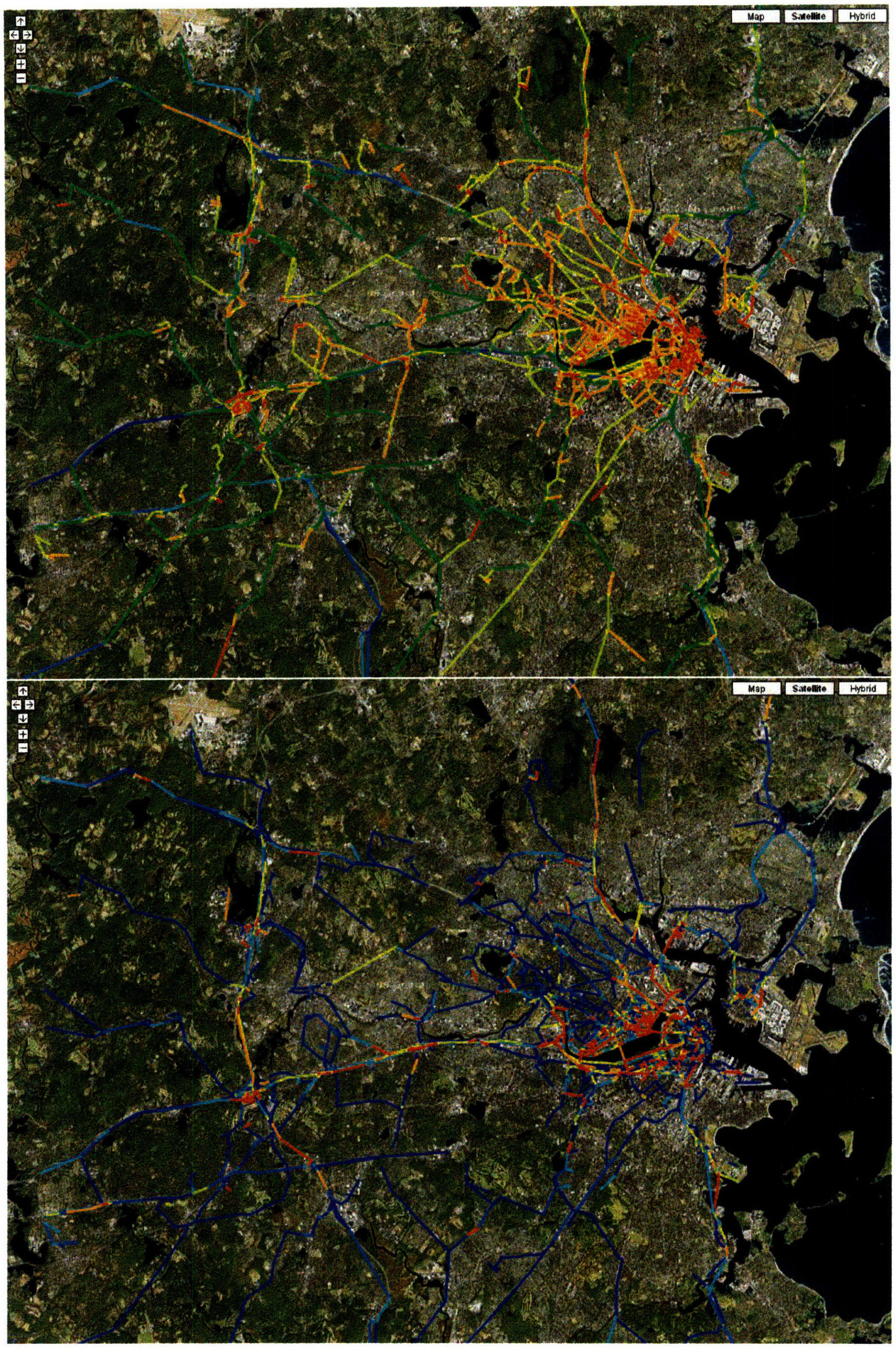
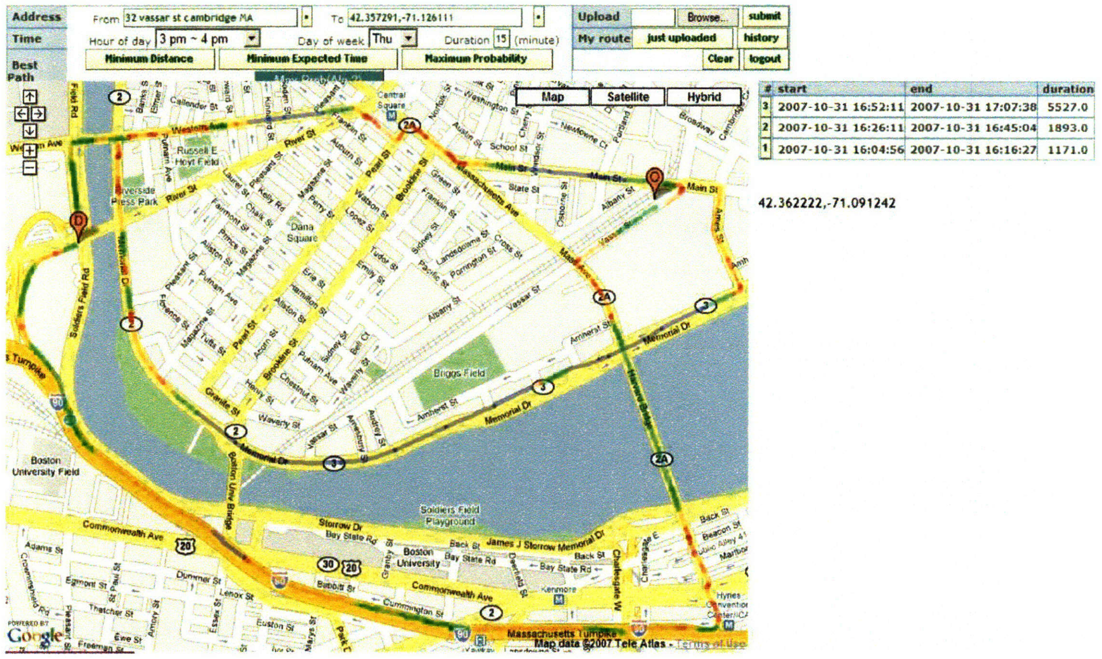


Figure 8-3: Hot Spot Visualization. The top figure shows the color-coded mean velocity of road segments. The bottom figure shows the color-coded ratio of the standard deviation and the mean of the travel time of road segments. Road segments indicated with the red color are hot spots.

8.4.4 User Route Upload and Analysis

"My Drive" is a private member area where users can upload their own GPS track-log files and can see their routes on the map color coded by speed. The file to be uploaded should be in a valid GPS Exchange (GPX) file format. Users can press "just uploaded" button to see the trace they just uploaded, or they can also view the summary of the entire GPS traces they uploaded by clicking the "history" button. While users can get useful analysis information about their drives, users' track logs help us develop a more accurate and robust traffic information system. Login is required to look up users' own GPS traces.



From: 32 vassar st cambridge MA
To: 42.357291,-71.126111
Hour of day: 3 pm ~ 4 pm
Day of week: Thu
Duration: 15 (minute)

#	start	end	duration
3	2007-10-31 16:52:11	2007-10-31 17:07:38	5527.0
2	2007-10-31 16:26:11	2007-10-31 16:45:04	1893.0
1	2007-10-31 16:04:56	2007-10-31 16:16:27	1171.0

42.362222,-71.091242

Figure 8-4: Example Interface Showing a User's Drives Color Coded by Speed. Red : 0∼10 mph, Orange : 10∼20 mph, Green : 20∼30 mph, Cyan : 30∼40 mph : Blue : over 40 mph

Chapter 9

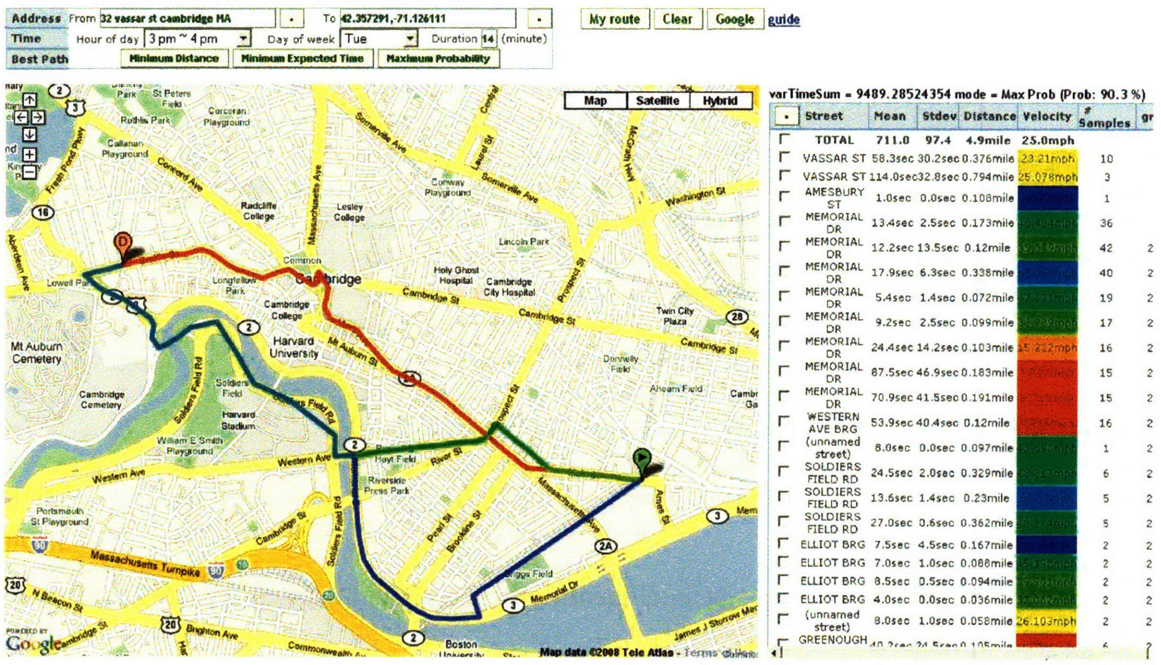
Experiments

In this chapter we evaluates our intelligent traffic information system by the field experiments.

9.1 Path Examples

Figure 9-1 shows different optimal paths from a start location (the green arrow) to a destination (the red "D"), according to three different criteria: the minimum distance route (this is the same route recommended by Google Maps and is indicated as the topmost red route), the minimum expected time route (the middle green route), and the maximum probability route with a deadline of 14 minutes (the bottom blue route). Our system estimates that minimum distance route (which is 3.1 miles) will take 18 minutes on a Tuesday afternoon. Our system's minimum expected time route (which is 3.5 miles) takes only 11 minutes and 45 seconds. The maximum probability route (which is 4.1 miles) takes 11 minutes and 51 seconds on average with 90.3% guarantee of arriving on time. The minimum distance route and the minimum expected time route have lower probabilities of 1% and 88.5%, respectively.

Figure 9-2 shows how the optimal paths change according to the time of day. For a travel from MIT to the Massachusetts turnpike Cambridge entrance, we queried both minimum expected time path and maximum arrival probability path for both 3$\sim$4 pm and 4$\sim$5 pm on Mondays. The minimum expected time path for 3$\sim$4 pm is the red line. The



varTimeSum = 9489.28524354 mode = Max Prob (Prob: 90.3 %)

	Street	Mean	Stdev	Distance	Velocity	# Samples	gr
	TOTAL	711.0	97.4	4.9mile	25.0mph		
	VASSAR ST	58.3sec	30.2sec	0.376mile	🟨 23.21mph	10	
	VASSAR ST	114.0sec	32.8sec	0.794mile	🟨 25.078mph	3	
	AMESBURY ST	1.0sec	0.0sec	0.108mile	🟦	1	
	MEMORIAL DR	13.4sec	2.5sec	0.173mile	🟩	36	
	MEMORIAL DR	12.2sec	13.5sec	0.12mile	🟩	42	2
	MEMORIAL DR	17.9sec	6.3sec	0.338mile	🟦	40	2
	MEMORIAL DR	5.4sec	1.4sec	0.072mile	🟩	19	2
	MEMORIAL DR	9.2sec	2.5sec	0.099mile	🟩	17	2
	MEMORIAL DR	24.4sec	14.2sec	0.103mile	🟧 15.222mph	16	2
	MEMORIAL DR	87.5sec	46.9sec	0.183mile	🟥	15	2
	MEMORIAL DR	70.9sec	41.5sec	0.191mile	🟥	15	2
	WESTERN AVE BRG	53.9sec	40.4sec	0.12mile	🟥	16	2
	(unnamed street)	8.0sec	0.0sec	0.097mile	🟩	1	2
	SOLDIERS FIELD RD	24.5sec	2.0sec	0.329mile	🟩	6	2
	SOLDIERS FIELD RD	13.6sec	1.4sec	0.23mile	🟦	5	2
	SOLDIERS FIELD RD	27.0sec	0.6sec	0.362mile	🟩	5	2
	ELLIOT BRG	7.5sec	4.5sec	0.167mile	🟦	2	2
	ELLIOT BRG	7.0sec	1.0sec	0.088mile	🟩	2	2
	ELLIOT BRG	8.5sec	0.5sec	0.094mile	🟩	2	2
	ELLIOT BRG	4.0sec	0.0sec	0.036mile	🟩	2	2
	(unnamed street)	8.0sec	1.0sec	0.058mile	🟨 26.103mph	2	2
	GREENOUGH	40.2sec	24.5sec	0.105mile	🟥	6	2

Figure 9-1: The User Interface to the Traffic Information System with Highlighted Paths and Travel Times Found by the Algorithms Described in Chapter 6.

maximum arrival probability path for 3∼4 pm is also the red line. The minimum expected time path for 4∼5 pm is the orange line. The maximum arrival probability path path for 4∼5 pm is the green line. The Google Maps give blue line. From this result, we can observe that it is better to use longer but faster path for the time window 3∼4 pm, which is not a rush hour, and it is better to avoid the Mass Ave bridge during the time window 4∼5 pm, when the traffic through the bridge starts to increase. The paths found by our system avoid the highly congested Massachusetts Avenue, which the path Google Maps provide uses, in rush hours as shown by the orange line in Figure 9-2.

9.2 Selecting a Trajectory

Identifying taxi trajectories that follow a specific sequence of road segments for a given route or an origin and destination pair is necessary for testing our system. A given route can be described as a sequence of segments. While scanning the segments of Map Matching Table, if the exact match to the sequence of the given route is found, we add the delays for

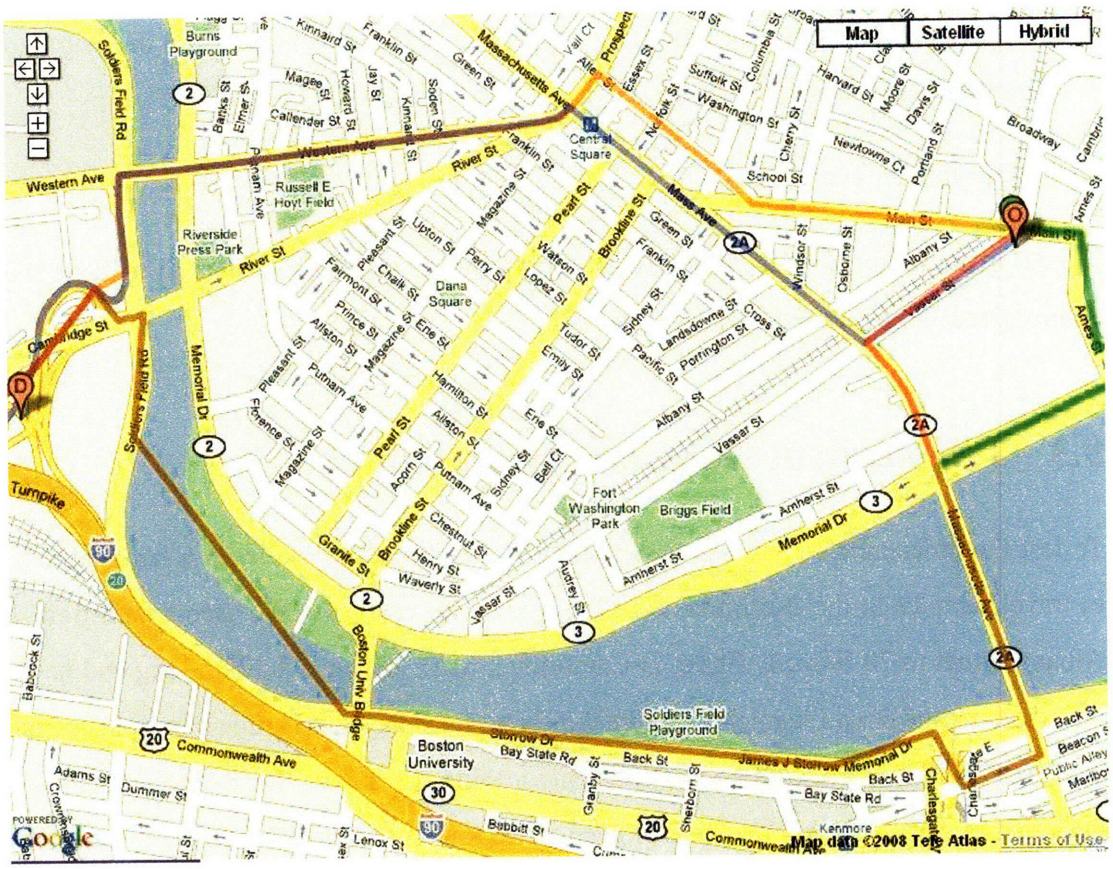


Figure 9-2: Optimal Path Query Result for Different Time Windows. Travel from MIT ("O") to Massachusetts turnpike entrance ("D").

the entire segments getting a travel time sample for the given route. However, for some road segments where map matching is not reliable, this kind of sequence matching does not work well. In that case, we find all the route that connects the origin and destination, and classify them by visualizing routes on a map. The method to obtain the origin-to-destination travel samples using the CarTel GPS data is described as follow:

1. Make small a square region around the origin and destination, respectively.

2. Scanning the GPS data samples of each car ordered by time, find a GPS point entering the origin region and remember the time when the car leaves the origin region (call it "time 1").

3. Check if the car enters the destination region in a given threshold time interval (the maximum possible time interval for the origin to destination travel).

4. If true, save the entering time (call it, “time 2”). The travel time from the origin to destination is “time 2” - “time 1”.

5. Repeat from 2 to 4.

We get many route samples with the described method, and we use them to validate the assumptions and to evaluate our system.

9.3 Independent Gaussian Assumption

To test the independent Gaussian assumption, we identified a route with a large number of travel samples in the Boston taxi CarTel trajectories database. We used a path from 1 Albany Street to 70 Waverly Street in Cambridge, MA , which is 1.12km long and has 5 intersections and 6 road segments. Figure 9-3 shows the delay distribution of road segments and total travel time distribution of the route. Table 9.1 shows the mean and standard deviation of each road group. The estimated travel time for the path based on the statistics of each road group is shown in the row starting with “Estimation”. The row starting with “Measurement” shows the measured statistics of the travel time for the path based on the 1886 trajectory samples.

Table 9.1: Comparison of the Estimated Mean and Standard Deviation with the Measured Mean and Standard Deviation for the Route from 1 Albany Street to 70 Waverly Street

Street	Mean (sec)	Stdev (sec)	Distance (miles)	Number of Samples
Group 1	18.7	4.9	0.16	4204
Group 2	20.9	18.5	0.117	5735
Group 3	27.6	23.4	0.034	5817
Group 4	13.3	3.6	0.098	5651
Group 5	23.6	6.9	0.214	3012
Group 6	35.7	16.1	0.264	2824
Estimation	139.8	35.2	0.9	
Measurement	141.8	43.9	0.9	1886

Table 9.2 shows that the percentiles of estimation and measurements are mostly in 5 second range. Figure 9-4 shows that the mean and standard deviation of estimation and measurement is similar for all the hour of day.

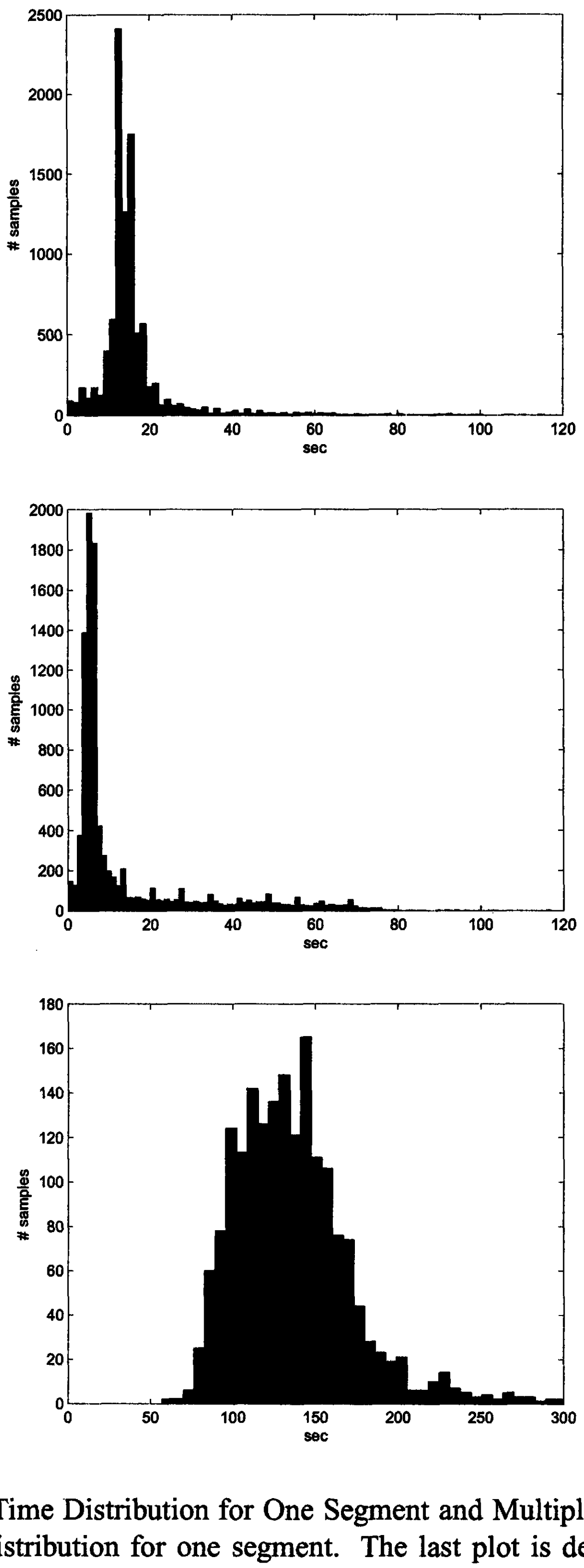


Figure 9-3: Delay Time Distribution for One Segment and Multiple Segments. The first two plot is delay distribution for one segment. The last plot is delay distribution for 6 segments including the first two segments

Table 9.2: Comparison of the Percentiles between Estimation and Measurement for the Route from 1 Albany Street to 70 Waverly Street

Percentile (%)	Estimation (sec)	Measurement (sec)
50	139.6	136.0
60	148.6	144.0
70	158.1	153.0
80	169.6	164.0
85	176.1	172.0
90	185.1	182.0
95	197.6	207.0
99	221.7	282.0

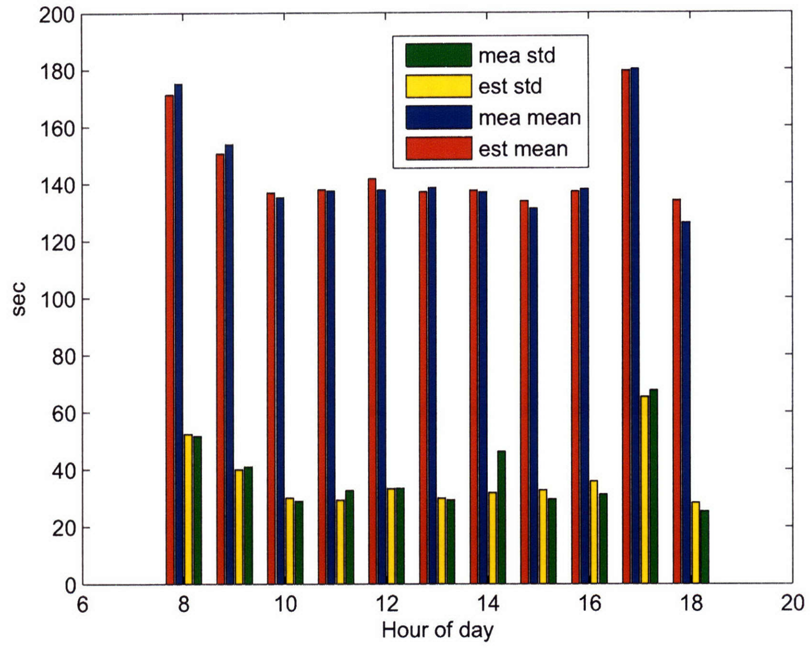


Figure 9-4: Comparison of the Mean and the Standard Deviation between Estimation and Measurement per Hour of Day for the Route from 1 Albany Street to 70 Waverly Street

Figure 9-5 shows that the estimated distribution by our system is very close to the real distribution.

More specifically, Figure 9-6 shows that the empirical data is very similar to the Gaussian distribution, especially in the probability interval from 0.05 to 0.95.

Thus, our assumptions will make sense for the stochastic planning for reaching the destination with the probability in this range. The discrepancies observed over 0.95 and

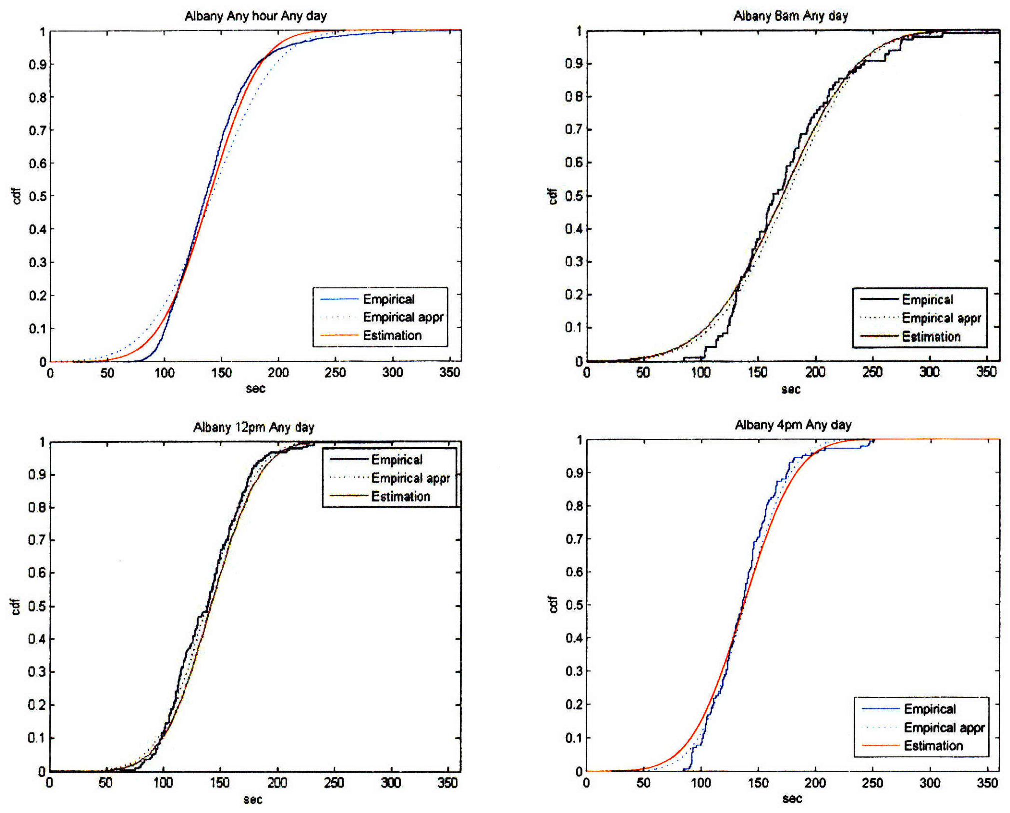
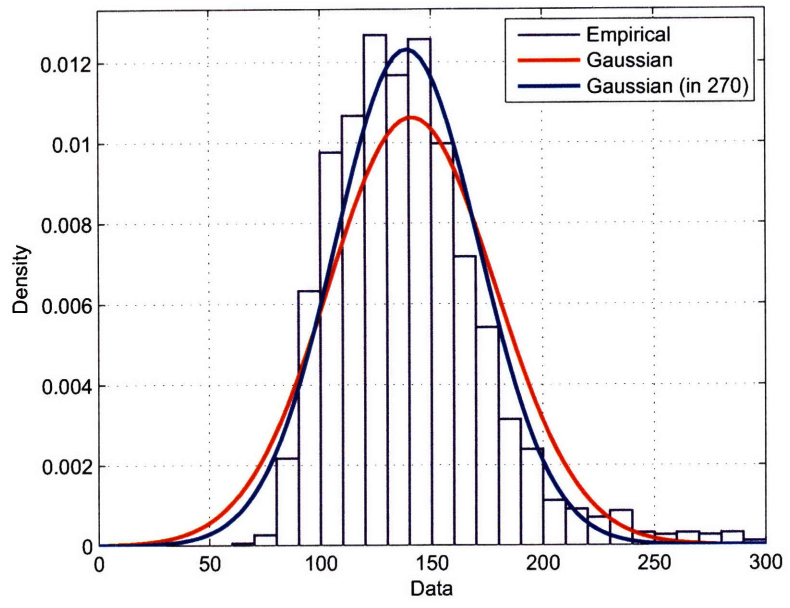


Figure 9-5: Comparison of CDF's between Estimation and Measurement for the Route from 1 Albany Street to 70 Waverly Street for Different Hours of Day

under 0.05 shows the limitation of our algorithms due to our assumption. For example, as shown in Figure 9-6 (Bottom), our system will estimate that the users can reach the goal with 99% probability if they leave 230 seconds before the deadline, but the empirical data shows that we will get only 97% chance, and if users want 99% guarantee they should leave about 270 seconds before the deadline. The discrepancy over 0.95 is caused by some unusual long delay, which might be due to unexpected events, construction work, or data gathering noise such as taxi drivers' intentional stops or slow drives. The discrepancy below 0.05 is due to the Gaussian distribution spans to the negative value whereas the travel time cannot be negative. We observe less discrepancy in case of Figure 9-7, which is using only the data from 1 $\sim$ 2 pm whereas Figure 9-6 are for entire hours. This result suggests that narrowing the data by conditions that affect the traffic delays such as time of day makes the delay distribution look more like independent Gaussian distribution. Thus, in our ongoing



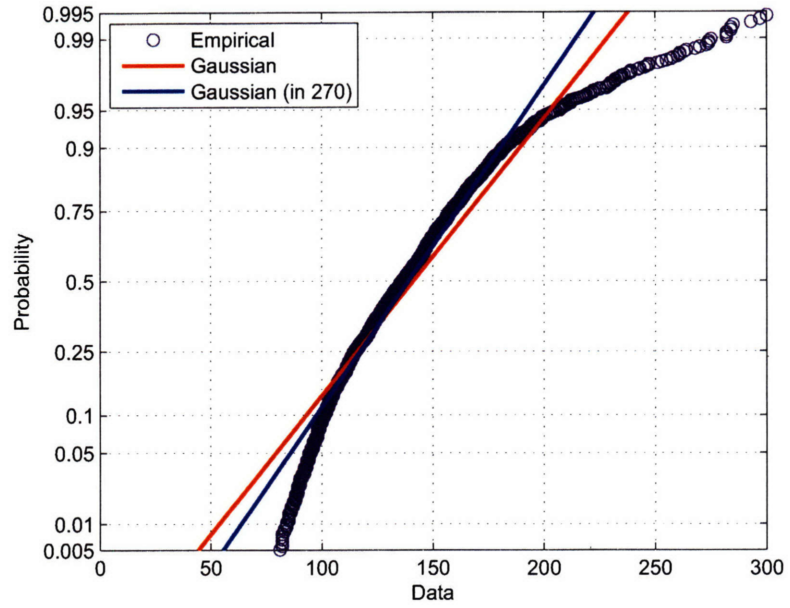
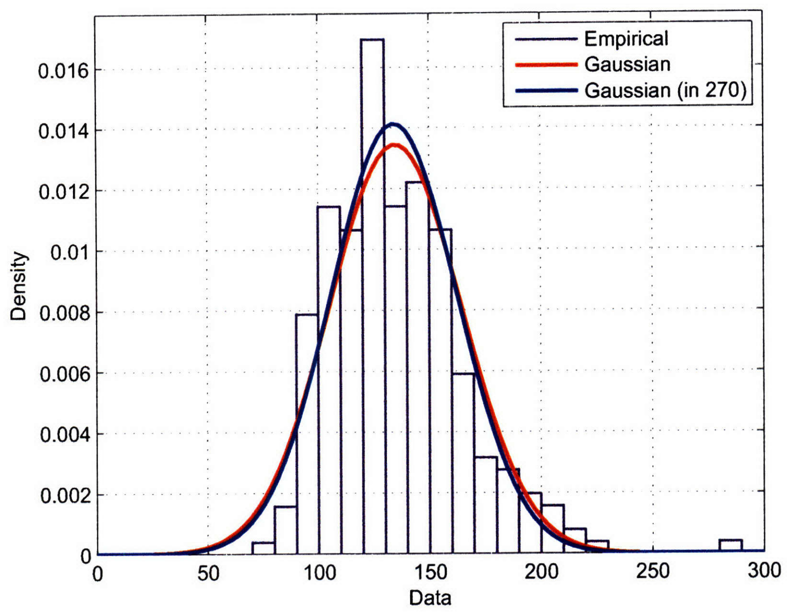


Figure 9-6: (Top) Histogram of Empirical Travel Time Data and Gaussian Fits for Any Time for Weekdays. (Bottom) Probability Comparison between Empirical Data and Gaussian Fits for Any Time for Weekdays, where the Y axis was scaled to make the Gaussian CDF linear. "Empirical" indicates the travel time measurement by driving, "Gaussian" indicates the Gaussian fit for the entire data and "Gaussian (in 270)" indicates the Gaussian fit using only the data in 270 seconds. The unit of the X axis is second.



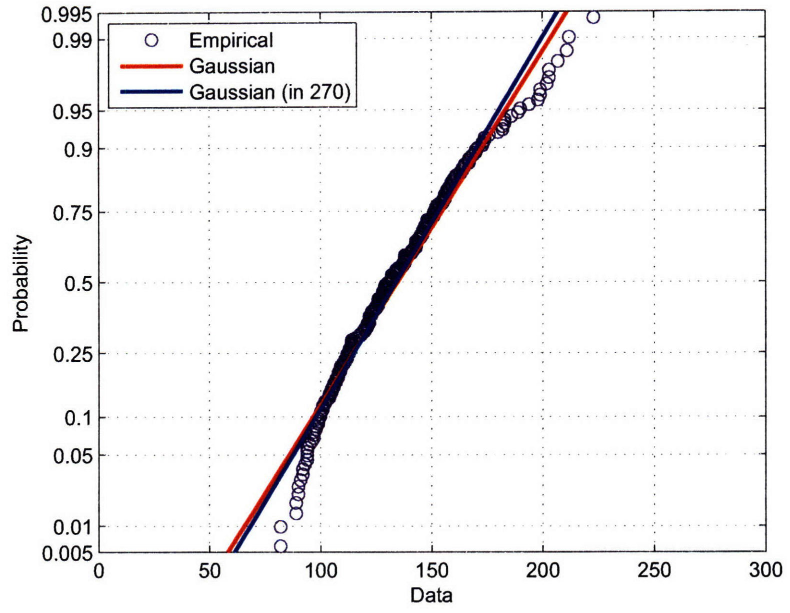


Figure 9-7: (Top) Histogram of Empirical Travel Time Data and Gaussian Fits for 1 $\sim$ 2 pm for Weekdays. (Bottom) Probability Comparison between Empirical Data and Gaussian Fits for 1 $\sim$ 2 pm for Weekdays, where the Y axis was scaled to make the Gaussian CDF linear. "Empirical" indicates the travel time measurement by driving, "Gaussian" indicates the Gaussian fit for the entire data and "Gaussian (in 270)" indicates the Gaussian fit using only the data in 270 seconds. The unit of the X axis is second.

research, we are investigating the proper conditions that constraints the traffic delays.

This observation provides some evidence that the independent Gaussian assumption used for Algorithm 6 and 7 holds for the Boston taxi CarTel database, but more testing on road segments with more associated travel data is necessary. As the database grows every day, we plan to continue this validation.

9.4 Overall Path Goodness

Four different routes from MIT to Boston Logan airport as shown in Figure 9-8 were examined using taxi paths and human test driving.

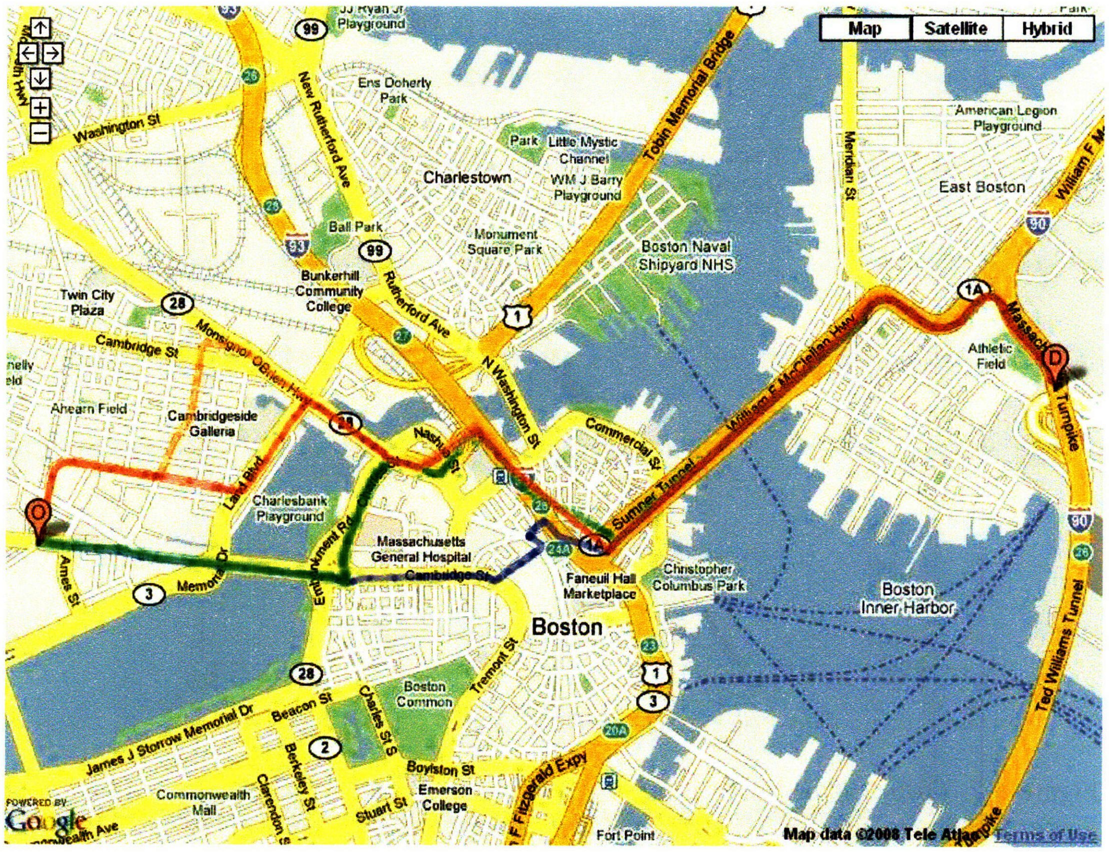


Figure 9-8: Four Major Alternative Routes from MIT ("O") to Boston Logan Airport ("D"). Red line : route 1, Orange line : route 2, Green line : route 3, Blue line : route 4

The estimated mean of each path during 7 am to 9 pm was 872 seconds, 899 seconds, 816 seconds, and 795 seconds for route 1 (6.9km), route 2 (7.2km), route 3 (6.7km), and route 4 (6.2km) respectively. The measured mean was 869 seconds, 895 seconds, 811 seconds, and 799 seconds. Thus, the estimated minimum expected time path, route 4, agrees with the measurement. Figure 9-9 gives the maximum probability path. The estimated probability is similar to the measured probability. From both estimation and measurement, for a deadline less than or equal to 12 minutes, the route 3 is the best, but for a deadline larger than 12 minutes the route 4 is the best. We can observe that route 4, which is the minimum expected time path is the worst path for a deadline less than 12 minutes.

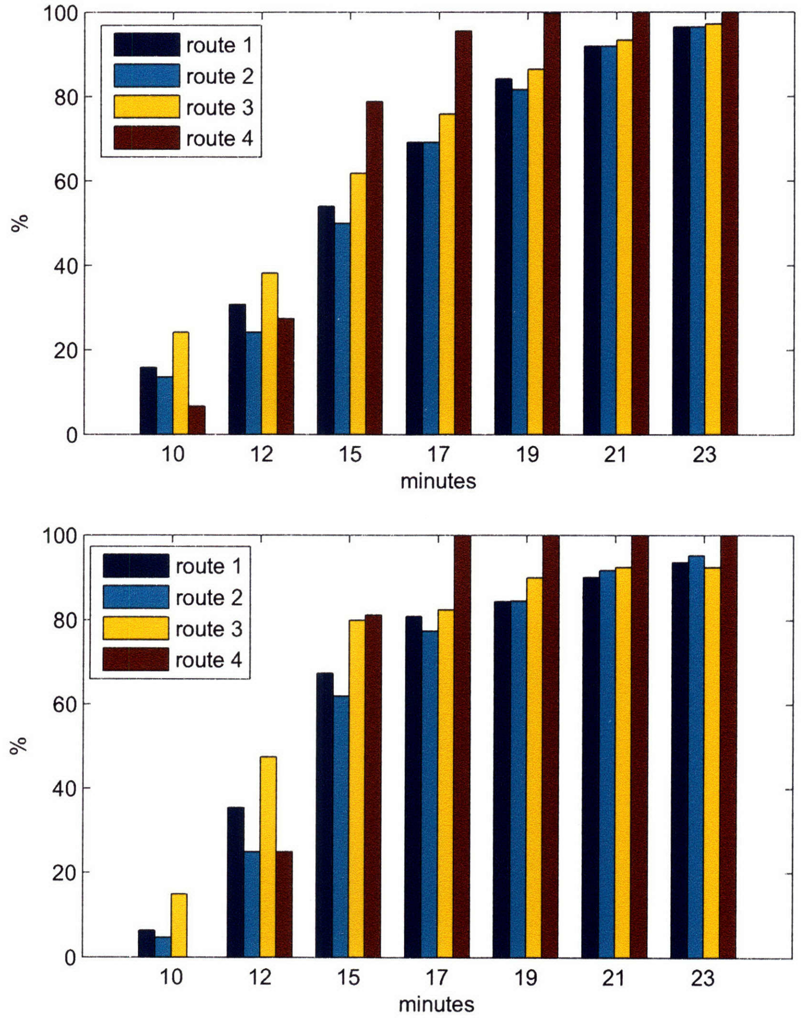


Figure 9-9: The Probability of Arriving in a Given Deadline from a Start location to a Destination for Different Times of Day. (Top: Estimation, Bottom: Measurement)

Chapter 10

Conclusion and Future Work

We developed efficient stochastic shortest path algorithms, and implemented a traffic information system using historical and current information. We evaluated the system with actual measured travel time for selected routes, and observed that our system's optimal path and travel time estimates are close to the reality. We envision that more accurate prediction and better navigation are possible when more data are gathered and when users share their travel time information with others through our system.

In the future, we are interested in developing path planning algorithms for multiple users. We are also interested in improving our algorithms and system by considering dependencies of each road segment and by using better modeling of delay distributions. We are currently extending the algorithms to integrate current traffic information with historical information to make more accurate estimation and to predict the future traffic conditions. Considering various conditions affecting traffic like weather, construction work, events is also part of our current plans. Finally, we plan to integrate this planning system with autonomous vehicles.

Bibliography

[1] Airsage, inc. `http://www.airsage.com/`.

[2] Gdf 3.0 documentation and manual. `http://www.ertico.com/en/links/links/gdf_-_geographic_data_files.htm#GDF_3.0_Documentation_&_Manual`.

[3] Google maps api. `http://code.google.com/apis/maps/`.

[4] Intellione technologies corporation. `http://www.intellione.com/`.

[5] Navteq data. `http://www.navteq.com/developer/index.html`.

[6] Tiger/line data. `http://www.census.gov/geo/www/tiger/`.

[7] M. Ben-Akiva, M. Bierlaire, H. Koutsopoulos, and R. Mishalani. Dynamit: a simulation based system for traffic prediction. In *DACCORD Short Term Forecasting Workshop*, 1998.

[8] D. Bernstein and A. Kornhauser. An introduction to map matching for personal navigation assistants. *New Jersey TIDE Center*, 1996.

[9] Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595., August 1991.

[10] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 853–864. VLDB Endowment, 2005.

[11] Hans Braxmeier, Volker Schmidt, and Evgueni Spodarev. Spatial extrapolation of anisotropic road traffic. *Image Anal Stereol*, 23:185–198, 2004.

[12] P. Carstensen. The complexity of some problems in parametric linear and combinatorial programming. *Ph.D. Thesis, Mathematics Dept., U. of Michigan, Ann Arbor, Mich.*, January 1983.

[13] I. Chabini. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. *Transportation Research Record*, 1645:170–175, 1998.

[14] Roland Chrobok, Joachim Wahle, and Michael Schreckenberg. Traffic forecast using simulations of large scale networks. In *IEEE Conference on Intelligent Transportation Systems*, Oakland, CA, USA, August 2001.

[15] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality af a*. *J. ACM*, 32(3):505–536, 1985.

[16] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[17] Bellman R. E. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[18] H. Frank. Shortest paths in probabilistic graphs. *Operations Research*, 17(4):583–599, 1969.

[19] Cyril Furtlehner, Jean-Marc Lasgouttes, and Arnaud de la Fortelle. A belief propagation approach to traffic prediction using probe vehicles. In *Intelligent Transportation Systems Conference*, pages 1022–1027, September 2007.

[20] JS Greenfeld. Matching gps observation to locations on a digital map. In *the 81th Annual Meeting of the Transportation Research Board*, 2002.

[21] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen K. Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. CarTel: A Distributed Mobile Sensor Computing System. In *4th ACM SenSys*, Boulder, CO, November 2006.

[22] Evangelos Kanoulas, Yang Du, Tian Xia, and Donghui Zhang. Finding fastest paths on a road network with speed patterns. In *the 22nd International Conference on Data Engineering*, pages 10– 10, April 2006.

[23] J.-S. Kim. Node based map matching algorithm for car navigation system. In *the International Symposium on Automotive Technology and Automation*, pages 121–126, 1996.

[24] Harris C.B. Wong R.V.C. Krakiwsky, E.J. A kalman lter for integrating dead reckoning, map matching and gps positioning. In *IEEE Position Location and Navigation Symposium*, pages 39–46, 1988.

[25] R-M Kung, E Hanson, Y Ioannidis, T Sellis, L Shapiro, and M Stonebraker. Heuristic search in database systems. In *Proceedings from the first international workshop on Expert database systems*, pages 537–548, Redwood City, CA, USA, 1986. Benjamin-Cummings Publishing Co., Inc.

[26] Wei-Hua Lin. A gaussian maximum likelihood formulation for short-term forecasting of traffic flow. In *IEEE Conference on Intelligent Transportation Systems*, Oakland, CA, USA, August 2001.

[27] R. Loui. Optimal paths in graphs with stochastic or multidimensional weights. *Communications of the ACM*, 26(9):670–676, 1983.

[28] Ronald Prescott Loui. Optimal paths in graphs with stochastic or multidimensional weights. *Commun. ACM*, 26(9):670–676, 1983.

[29] I. Murthy and S. Sarkar. Exact algorithms for the stochastic shortest path problem with a decreasing deadline utility function. *European Journal of Operational Research*, 103:209–229, 1997.

[30] Maan E. El Najjar and Philippe Bonnifait. A road-matching method for precise vehicle localization using belief theory and kalman filtering. *Journal Autonomous Robots*, 19(2):173–191, September 2005.

[31] Evdokia Nikolova, Matthew Brand, and David Karger. Optimal route planning under uncertainty. In *International Conference on Automated Planning and Scheduling*, 2006.

[32] Evdokia Nikolova, Jonathan A. Kelner, Matthew Brand, and Michael Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *ESA*, pages 552–563, 2006.

[33] Moshe E. Ben-Akiva Qi Yang, Haris N. Koutsopoulos. Simulation laboratory for evaluating dynamic traffic management systems. *Transportation Research Record*, 1710:122–130, 2000.

[34] Mohammed A. Quddus, Washington Yotto Ochieng, Lin Zhao, and Robert B. Noland. A general map matching algorithm for transport telematics applications. *GPS Solutions Journal*, 7(3):157167, 2000.

[35] Kumud K. Sanwal and Jean Walrand. Vehicles as probes. *Technical Report UCB-ITS-PWP-95-11, California Partners for Advanced Transit and Highways (PATH)*, January 1995.

[36] D. Schrank and T. Lomax. The 2007 urban mobility report. *Annual report, Texas Transportation Institute, The Texas A&M University System*, September 2007.

[37] Drane C.R. Scott, C.A. Increased accuracy of motor vehicle position estimation by utilizing map data, vehicle dynamics and other information sources. In *the Vehicle Navigation and Information Systems Conference*, pages 585–590, 1994.

[38] C. E. Sigal, A. A. B. Pritsker, and J. J. Solberg. The stochastic shortest route problem. *Operations Research*, 28(5):1122–1129, 1980.

[39] J. Tanaka. Navigation system with map-matching method. In *the SAE International Congress and Exposition*, pages 45–50, 1990.

[40] M. Quddus W. Y. Ochieng and R. B. Noland. Map-matching in complex urban road networks. 2003.

[41] Michael P. Wellman, Matthew Ford, and Kenneth Larson. Path planning under time-dependent uncertainty. In *11th Conference on Uncertainty in Artificial Intelligence*, pages 532–539, August 1995.

[42] Christopher E. White, David Bernstein, and Alain L. Kornhauser. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies*, 8(1-6):91–108, February-December 2000.

[43] Haichen Xu and Daniel J. Dailey. Real time highway traffic simulation and prediction using inductance loop data. In *Vehicle Navigation and Information Systems Conference*, Seattle, WA, USA, July 1995.

[44] Ramin Yasdi. Prediction of road traffic using a neural network approach. *Neural Computing and Applications*, 8(2):135–142, 1999.

[45] Jungkeun Yoon, Brian Noble, and Mingyan Liu. Surface street traffic estimation. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 220–232, New York, NY, USA, 2007. ACM.