Congestion-Aware Traffic Routing for Large-Scale Mobile Agent Systems

by

Sejoon Lim

B.S., Seoul National University (2002)
S.M., Massachusetts Institute of Technology (2008)

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

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2012

Author ....
Department of Electrical Engineering and Computer Science
June 29, 2012

Certified by ....
Daniela Rus
Professor
Thesis Supervisor

Accepted by ....
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Students

Congestion-Aware Traffic Routing for Large-Scale Mobile Agent Systems

by

Sejoon Lim

B.S., Seoul National University (2002)

S.M., Massachusetts Institute of Technology (2008)

Submitted to the Department of Electrical Engineering and Computer Science
on June 29, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Traffic congestion is a serious world-wide problem. Drivers have little knowledge of historical and real-time traffic congestion for the paths they take and often tend to drive suboptimal routes. Congestion phenomena are sure to be influenced by the coming of autonomous cars. This thesis presents route planning algorithms and a system for either autonomous or human-driven cars in road networks dealing with travel time uncertainty and congestion.

First, a stochastic route planning algorithm is presented that finds the best path for a group of multiple agents. Our algorithm provides mobile agents with optimized routes to achieve time-critical goals. Optimal selections of agent and visit locations are determined to guarantee the highest probability of task achievement while dealing with uncertainty of travel time. Furthermore, we present an efficient approximation algorithm for stochastic route planning based on pre-computed data for stochastic networks.

Second, we develop a distributed congestion-aware multi-agent path planning algorithm that achieves the *social optimum*, minimizing aggregate travel time of all the agents in the system. As the number of agents grows, congestion created by agents' path choices should be considered. Using a data-driven congestion model that describes the travel time as a function of the number of agents on a road segment, we develop a practical method for determining the optimal paths for all the agents in the system to achieve the social optimum. Our algorithm uses localized information and computes the paths in a distributed manner. We implement the algorithm in multi-core computers and demonstrate that the algorithm has a good scalability.

Third, a path planning system using traffic sensor data is then implemented. We predict the traffic speed and flow for each location from a large set of sensor data collected from roving taxis and inductive loop detectors. Our system uses a data-driven traffic model that captures important traffic patterns and conditions using the two sources of data. We evaluate the system using a rich set of GPS traces from 16,000 taxis in Singapore and show that the city-scale congestion can be mitigated by planning drivers' routes, while incorporating the congestion effects generated by their route choices.

Thesis Supervisor: Daniela Rus
Title: Professor

Acknowledgments

I have many people and organizations to acknowledge and thank for their support throughout my doctorate program. I would like to express my deep gratitude to my advisor, Prof. Daniela Rus, for her support and guidance during my time at MIT. I would not have been able to finish my thesis without her guidance and help.

I am also very grateful to my committee, Prof. Hari Balakrishnan and Prof. Sam Madden, for their suggestions and guidance from the start of my graduate study.

I thank all the collaborators including Prof. Javed Aslam, Prof. Jakob Eriksson, Prof. David Gifford, Prof. Evdokia Nokilova, Christian Sommer, and Xinghao Pan.

Thanks are also due to all the members of the Distributed Robotics Laboratory, both past and present, for helpful discussions and encouragement.

I am also grateful to all the data providers including CarTel team at MIT, Land Transportation Authority in Singapore, and ComfortDelGro in Singapore.

I would also like to thank the various funding organizations and companies which have funded my research. These funding sources include the Singapore-MIT Alliance for Research and Technology (The Future of Urban Mobility project), the NSF (grant numbers CPS-0931550, 0735953), the ONR (grant numbers N00014-09-1-105, N00014-09-1-1031), and the Samsung Scholarship.

Finally, above all, I would like to thank my family – my mother, father, brother, and especially my wife, Grace Lee. They have been infinitely patient with me and have encouraged me throughout my entire PhD process. I would not have made it this far without their love and support through all these years, and for that I am eternally grateful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Traffic congestion is clearly a serious problem: a recent survey estimates that the annual nationwide cost of traffic congestion in the US was $101 billion in 2010, including 4.8 billion hours in lost time and 1.9 billion gallons in wasted fuel. This is an average of $713 per auto commuter [67]. Drivers today have little knowledge of historical and real-time traffic congestion for the paths they take, and even when they do (e.g., from "live" traffic updates), there are limited means for using that information to find good paths. As a result, drivers often tend to drive suboptimal routes, and time is used inefficiently when they leave well in advance to make an important deadline. Recent technology developments are so promising that autonomous cars will be prevalent in the near future [125, 91], and congestion phenomena are sure to be influenced by the coming of autonomous cars. Such cars will need a motion planning algorithm to navigate from A to B.

In this thesis, our goal is to solve city-scale congestion problems by understanding the traffic conditions and using them for planning drivers' paths. Our path planning system is intended to find driving directions that lead to fast and reliable travel times for individual drivers and less congestion on a city scale.

Current in-car navigation systems or Web-based route planning systems provide a driver a path from his origin to destination. However, oftentimes the path turns out to be congested. Avoiding congestion and finding a good path are not trivial, and the challenge is two-fold: (1) it is hard to predict the travel time for a road segment since many factors such as road conditions, congestion, and events affect the travel time, and (2) congestion is not fixed but rather is an aggregate effect of drivers' choice of road segments to take for their trips. Thus, a driver's path choice should be optimized considering the other drivers' path choices, and the decision about paths affects other drivers' travel time at the same time.

The challenges addressed in this thesis are: Can we optimize drivers' paths under uncertainty and congestion? Can we reduce city-scale congestion by finding better paths for individual drivers? There are several optimization criteria we consider. From the point of view of the individual we wish to maximize the probability of arriving at the destination and the amount an individual spends in traffic. From the point of view of a group of individuals (where the group can be viewed as city-scale traffic) we wish to minimize the overall time spent by all the users in traffic.

We address these questions using a data-driven approach. We collect GPS data using probe vehicles such as fleets of commercial vehicles, taxis, and individual drivers with GPS-enabled smart phones. When the GPS points are collected with high frequency (i.e. several points each minute) the entire trajectory of individual users can be recovered with great confidence. Traffic delay or speed information can be calculated from sequences of GPS trajectories.

In addition to GPS data, we also use data from inductive loop detectors. Inductive loop detectors are installed in the road pavement and detect the cars passing over them by detecting the change in electro-magnetic fields. The loop detector data is important because it gives us an accurate (ground truth) measure for all traffic at the locations where the loops are installed. Probe data and inductive loop data are complimentary. Inductive loop data gives us a measure of overall traffic at set points (where they are installed) but does not allow us to track individual drivers as they move through the network. GPS data allows us to track the drivers but presents information only about the GPS-enabled vehicles in the system instead of the overall traffic network.

The availability of traffic data from loop sensors and GPS probes provides a great opportunity for developing more intelligent solutions to the transportation problem. However, the following questions need to be addressed in order to understand the scope of the data: How can we turn available traffic data into a useful form? How can we develop a practical algorithm that use these data and find a good path under uncertainty and congestion? In this thesis, we develop practical algorithms for these questions and their implementation in a city-scale environment.

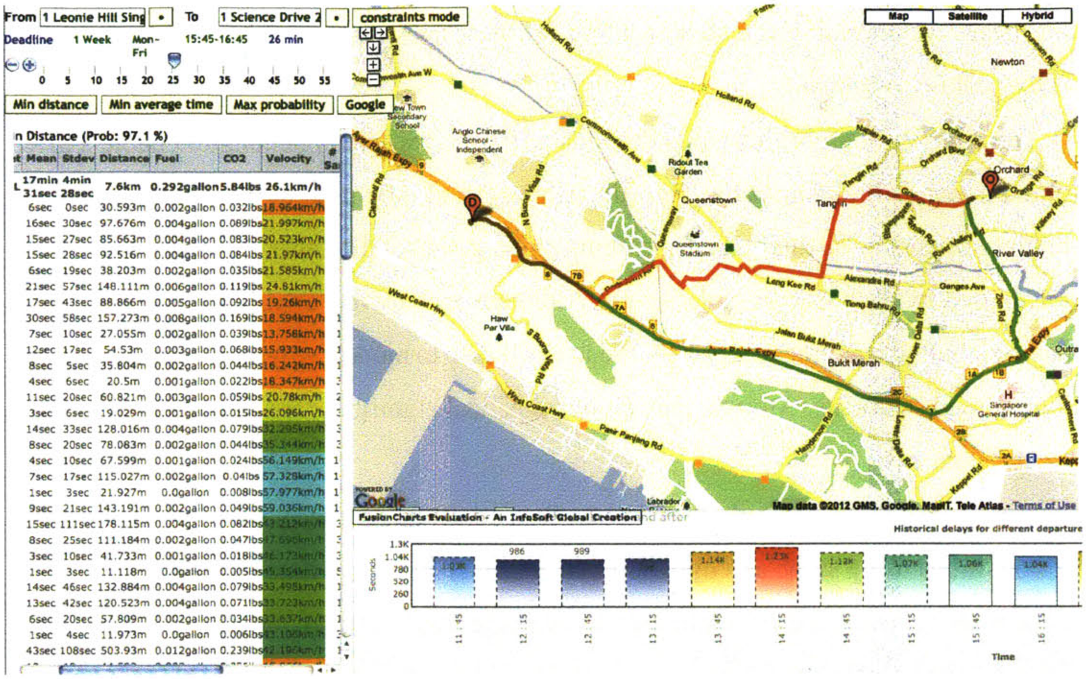
Our first set of algorithms includes stochastic shortest-path planning to enable a driver to find a fast and reliable path. This algorithms can also be used to coordinate a group of multiple drivers or agents. Our algorithms find a good solution that considers the congestion caused by each drivers path choice. Although each drivers contribution to congestion may be small, the aggregate is large. We use real data to model the uncertainty and congestion on road segments. More details on our practical algorithms are explained in Sections 1.1, 1.2, and 1.3.

Our second set of algorithms addresses data analysis, inference, and estimates for traffic conditions using static and dynamic sensors. We enable real-time estimation of flow on roads with dynamic sensors calibrated by existing static sensors. Based on the real-time flow and speed information that we obtain from dynamic sensors, we predict the congestion levels, and we make it possible to identify the current operational state of the roads. These estimates are very valuable information for both individual drivers and city planners. Details for data-driven traffic estimation and congestion detection are explained in Sections 1.4 and 1.5.

Finally, we implement these algorithms using data from a large set of taxi probes and loop detectors and demonstrate that the resulting traffic routing system can be used for guiding human-driven or autonomous cars. Our fast computation techniques and scalable design enable planning for the larger systems in which our algorithms will function. Section 1.6 provides more detail.

1.1 Route Planning on Stochastic Network

We describe a stochastic motion planning algorithm for multiple agents over a road network with stochastic travel cost, in the presence of visit node constraints. Stochastic travel cost models the uncertainty in travel cost depending on traffic conditions. The travel cost can be time, gas, money, or any combination thereof. Visit node constraints model a sequence of the agents' waypoints required to achieve a specific goal. Waypoints can model the



From 1 Leonie Hill Sing To 1 Science Drive 2 constraints mode

Deadline 1 Week Mon-Fri 15:45-16:45 26 min

Min distance | Min average time | Max probability | Google

n Distance (Prob: 97.1 %)

Mean	Stdev	Distance	Fuel	CO2	Velocity
17min 31sec	4min 28sec	7.6km	0.292gallon	5.84lbs	26.1km/h
6sec	0sec	30.593m	0.002gallon	0.032lbs	18.964km/h
16sec	30sec	97.676m	0.004gallon	0.089lbs	21.997km/h
15sec	27sec	85.663m	0.004gallon	0.083lbs	20.523km/h
15sec	28sec	92.516m	0.004gallon	0.084lbs	21.97km/h
6sec	19sec	38.203m	0.002gallon	0.035lbs	21.585km/h
21sec	57sec	148.111m	0.006gallon	0.119lbs	24.81km/h
17sec	43sec	88.866m	0.005gallon	0.092lbs	19.26km/h
30sec	58sec	157.273m	0.008gallon	0.169lbs	18.594km/h
7sec	10sec	27.055m	0.002gallon	0.039lbs	13.758km/h
12sec	17sec	54.53m	0.003gallon	0.068lbs	15.933km/h
8sec	5sec	35.804m	0.002gallon	0.044lbs	16.242km/h
4sec	6sec	20.5m	0.001gallon	0.022lbs	18.347km/h
11sec	20sec	60.821m	0.003gallon	0.059lbs	20.78km/h
3sec	6sec	19.029m	0.001gallon	0.015lbs	26.096km/h
14sec	33sec	128.016m	0.004gallon	0.079lbs	32.295km/h
8sec	20sec	78.083m	0.002gallon	0.044lbs	35.344km/h
4sec	10sec	67.599m	0.001gallon	0.024lbs	56.149km/h
7sec	17sec	115.027m	0.002gallon	0.04lbs	57.328km/h
1sec	3sec	21.927m	0.0gallon	0.008lbs	57.977km/h
9sec	21sec	143.191m	0.002gallon	0.049lbs	59.036km/h
15sec	111sec	178.115m	0.004gallon	0.082lbs	43.212km/h
8sec	25sec	111.184m	0.002gallon	0.047lbs	47.690km/h
3sec	10sec	41.733m	0.001gallon	0.018lbs	46.172km/h
1sec	3sec	11.118m	0.0gallon	0.005lbs	45.354km/h
14sec	46sec	132.884m	0.004gallon	0.079lbs	33.498km/h
13sec	42sec	120.523m	0.004gallon	0.071lbs	33.722km/h
6sec	20sec	57.809m	0.002gallon	0.034lbs	33.637km/h
1sec	4sec	11.973m	0.0gallon	0.006lbs	43.106km/h
43sec	108sec	503.93m	0.012gallon	0.239lbs	42.190km/h

Figure 1-1: Web interface for our Web-based optimal route query system. The interface gets user input through the top-left panel including address, time of travel, day of travel, and a deadline, if any. The red and green lines on the map show two different optimal paths using our system's statistics for the travel time estimates. The left panel shows the information of each segment in the routes, and the bottom panel shows the travel time estimates of early and late departure by multiples of 30 minutes from the user's intended departure time.

locations of resources to pick up or the location of one or multiple destinations.

Suppose a person calls a taxi company for help to get to any nearby hospital within ten minutes. How can the probability of getting there on time be maximized? It is important that a specific taxi, a specific hospital, and a specific path from the taxi's original location to you and from you to the hospital should be selected in order to maximize the probability. Similarly, if there is a fire in a building, how do we maximize the probability that at least one of the fire trucks can get water from one of several fire hydrants and get to the burning building within five minutes? A specific fire truck, a specific fire hydrant, and a specific path that connects the truck, the hydrant, and the building should be selected in order to maximize the probability.

The need for congestion-aware routing for complicated goals as illustrated above is in-

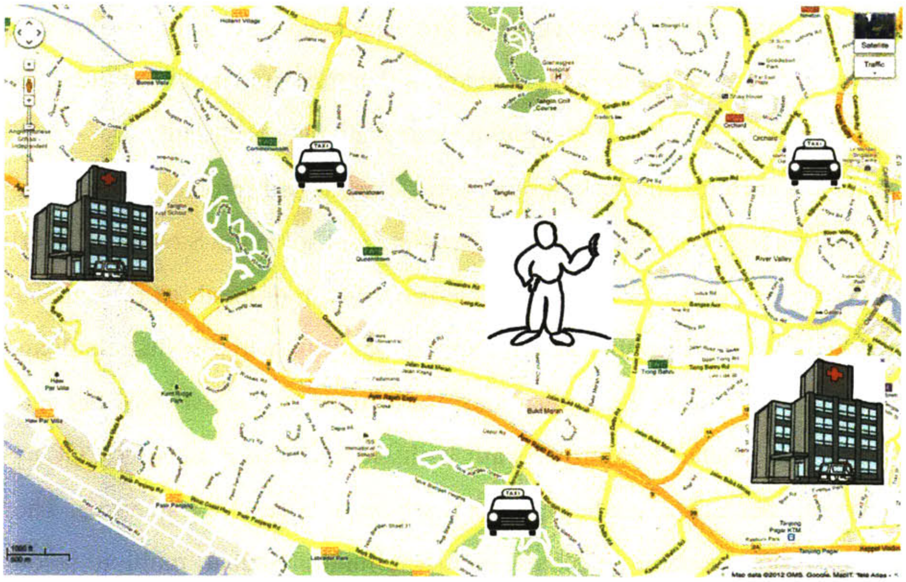


Figure 1-2: Example application: Taxi booking considering the taxi's location and uncertainty of travel time. The person want to go to one of the hospitals in ten minutes. There are multiple available taxis at different locations. Which taxi should be dispatched and which route should be used to get him to the hospital on time with the highest probability?

creasing. Today, with the technical achievements of advanced driver assistance systems, location-aware mobile networks, and acquisition of traffic information, it is more possible than ever before to calculate and provide the optimal path for either human-driven or autonomous vehicles.

We formulate a general framework for the stochastic shortest-path problem with visit node constraints to capture a large range of stochastic shortest-path problems in the presence of multiple agents, multiple resources, and multiple destinations.

The first challenge in solving the stochastic shortest-path problem with visit node constraints is that the optimal substructure does not hold [83]. In many other problems, the optimal substructure is used to develop efficient algorithms. In the hospital example, if the travel time of each road segment were a fixed deterministic value, the fastest vehicle to the patient would be guaranteed to be the fastest vehicle arriving at the hospital. In addition, the fastest path from the taxi's original location to the patient's location would consist of the fastest path from the taxi's original location to the hospital via the patient's location. However, this property does not hold for goals such as maximizing the arrival probability in a road network whose road segments are associated with a travel time distribution rather than a fixed value. The second challenge is that the number of possible routes grows exponentially with the number of waypoints describing the constraints.

We formulate different classes of stochastic shortest path problems depending on the structure of visit nodes constraints. First, we define and solve the Stochastic Shortest Path with Fixed Node Sequence (SSPFNS) problem, where an ordered list of nodes to be visited is specified. We provide an efficient method that deals with the first challenge. Many multiple-hop stochastic routing applications are formulated by SSPFNS. Second, we define and solve the Stochastic Shortest Path with General Constraints (SSPGC) problem, where constraints describe complicated combinations of node visit sequences. The SSPGC problem covers more general problems, including multiple-agent, multiple-resource, and multi-destination routing applications.

This research is motivated by motion planning in a road network, but the algorithm is general so that it can be applied to broader areas of robot motion planning in uncertain environments. For example, we envision that the algorithm can be extended to find a trajectory for a robot arm to move in a space maximizing the probability of achieving the goal within a given cost.

1.2 Stochastic Route Planning using Distance Oracles

This thesis presents and evaluates a new practical algorithm for planning the motion of vehicles (autonomous or human-driven) on roadways in the presence of uncertainty in traffic delays [86]. We build on prior work on stochastic motion planning [83, 101, 102], which introduces *stochastic motion planning* and route planning algorithms. This prior work presents novel provably correct algorithms that are very well suited for planning the motion of a vehicle on small-scale graphs; however, the computational complexity of this prior suite of stochastic motion planning algorithms makes these algorithms impractical to use for real-time motion planning on city-scale road networks.

The stochastic motion planning model incorporates traffic history in the form of probability distributions of delays observed on actual road segments. The probability distributions capture the historical delay observations that are assigned as weights on the edges of the road network. The stochastic motion planning algorithm minimizes a user-specified cost function of the delay distribution. Current route planning methods can produce routes with shortest *expected* time (based on historical traffic data) and some recent commercial systems, using almost-real-time data based on cell-phone locations [126, 116], can also compute “smartest” routes in dynamic traffic scenarios. The algorithm in this thesis improves significantly on this previous work by enabling fast routes guaranteed to maximize the likelihood of arriving at a destination within a specified deadline in city-scale route graphs, in the presence of limited data storage. This work provides a planning system that can be used by autonomous as well as human-driven vehicles for traffic routing applications that meet travel goals such as “when should you leave, and what path should you take, to reach the destination by the given deadline with high probability?” Preserving the inherently non-deterministic nature of actual traffic, we work with accurate probabilistic models of traffic-intense areas; in our work, we actually build a probabilistic model of an entire city’s road network based on a large set of heterogenous GPS traces.

More specifically, the algorithm in this thesis improves the state of the art on stochastic motion planning [101, 102] and its application to traffic routing [83]. The running time of the state-of-the-art stochastic-shortest-path algorithm is approximately quadratic in the number of nodes, rendering it too slow for real-time performance on city-scale networks. We improve upon their running time by using efficient data structures. The preprocessing algorithm runs in almost linear time and it does *not* depend on the origin and destination pair. The query algorithm runs in *sublinear* time (whereas the algorithms in [83] run in polynomial time). The running time of the query algorithm is almost *logarithmic*, offering exponential speedups over previous algorithms and allowing for almost instantaneous response times.

We implement the algorithms described in this thesis and packaged them as a route planning system. We evaluate the system using the road map of an entire city. Using GPS traces from a very large fleet of taxis (16,000 vehicles), we are able to extract a time-dependent, probabilistic model for each individual road segment. Our algorithms preprocess the road network equipped with probabilistic edge lengths into a data structure.

Using this data structure, our query algorithm can efficiently answer approximate stochastic shortest-path queries. Our experiments validate the theoretical performance predictions, showing improvements upon existing query algorithms by several orders of magnitude.

In summary, we propose an algorithm that improves upon the state of the art in stochastic motion planning, providing a method that can answer stochastic shortest-path queries in almost logarithmic time using a data structure that occupies space roughly proportional to the size of the network. Our method has proven worst-case guarantees and it is applicable to real-world scenarios. Applied to traffic navigation, we obtain a method that can efficiently find a path with almost maximum arrival probability given a user-specified deadline.

1.3 Multi-Agent Route Planning

Several of our recent traffic planning algorithms consider optimizing a driver's path in the presence of congestion [82, 83, 85]. However, as the number of drivers who use the system increases, the paths computed by an agent-centric system have degraded performance. Instead of optimizing individual travel times, we focus on optimizing the collective performance of a multi-agent transportation system by aiming to achieve the social optimum. The social optimum is defined as the minimum aggregated travel time for all the agents in the system. The social optimum approach is different from computing the user equilibrium to achieve the optimal path for each individual driver [127, 114]. In a social optimum planning system, some agents may be guided along expensive sub-optimal paths in order to achieve the overall best system performance.

In this thesis we describe a new distributed algorithm for planning paths for multiple agents in a road network. The goal is to minimize congestion and achieve the social optimum for traffic flow in the system. The social optimum minimizes the sum of the total cost for all the agents in the system. We wish to provide each agent with a decision system for selecting its path from origin to destination by making local decisions that depend only on the other agents whose paths might overlap in segments with this agent's path. This "localized" reasoning will ensure scalability and real-time performance.

Applications of this algorithm include route planning for transportation where the cars

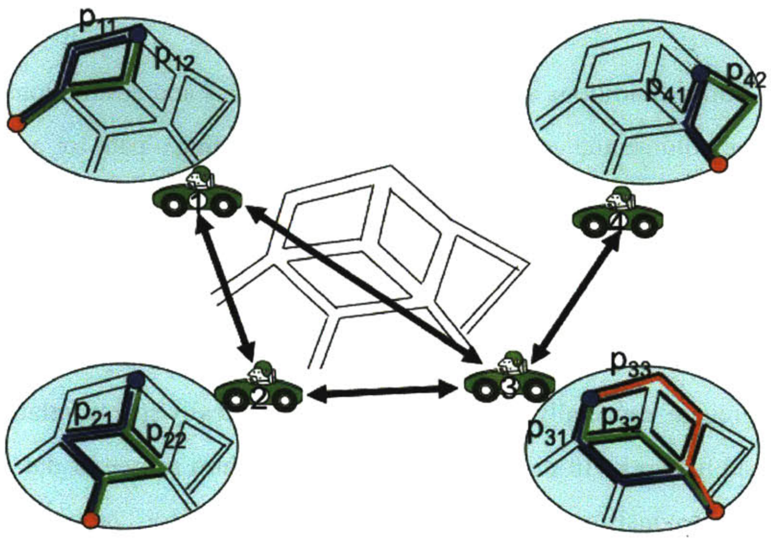


Figure 1-3: Distributed agent-centric approach. The neighbors of an agent are any agents whose potential paths overlap with those of the agent in any road segments. The potential paths for each agent is drawn in a blue circle. A pair of neighboring agents are indicated by a bi-directional arrow. Each agent maintains the probabilistic path decision vector, and updates it based on the neighbors' path decision vectors. The communication of path decision vectors is required only between neighbors.

are either driven by people or autonomous. Mitigating congestion in road networks is one of the most important challenges of urban transportation and multi-agent management of navigation. Our algorithm also benefits swarm robotics applications where many robots move in parallel. Mitigating collisions is a costly operation (with respect to perception, computation, and communication) and if the robots can organize their paths from their origins to their destinations to "spread" and minimize the number of likely intersection points, the overall system performance increases. Other applications include planning for evacuation and disaster response.

In our previous work [82, 83], we considered stochastic motion planning for a single agent from a given origin to a destination in a stochastic graph whose weights were probability distributions corresponding to historical congestion. In this thesis we extend this work to motion planning for multiple agents in stochastic graphs with a guaranteed social optimum for the solution.

1.4 Data-Driven Traffic Modeling and Estimation

Understanding traffic conditions and patterns, such as origins and destinations or trips, car routes, and traffic volume and congestion, is critical for finding a path for a driver and for urban traffic planning. Traffic information is collected today through manually conducted surveys (for origins, destinations, routes), or using static sensors such as traffic cameras and loop detectors[1] (for volume, congestion). However, survey data are often incomplete, inaccurate, and out of date, and static sensor data are incomplete and often difficult to analyze and aggregate, especially in real time. It is also typically sparse, relative to the size of the entire road network. In this thesis, we consider a third source of data: a vehicular sensor network that consists of a roving fleet of dynamic sensor "probes." Commercial vehicles are often outfitted with GPS devices that log their locations and speeds at regular intervals; increasingly, so are federal, state, and municipal vehicles. These vehicles form a mobile sensor network, providing real-time information on the state of the road network.

In a large-scale study conducted in the country of Singapore, we collected the travel data (including GPS, speed, and car state) from a fleet of 16,000 taxis for the month of August 2010, representing approximately 500 million individual data points. The taxis transmit the data using a cellular network. Thus, their data can be used in a real-time mode or in a historical mode.

We develop a method to estimate traffic volume for a road segment using taxi data and inductive loop detector data. Our model and inference procedures can be used to analyze traffic patterns and conditions from historical data, as well as to infer current patterns and conditions from data collected in real time. As such, our techniques provide a powerful new data source for traffic visualization, analysis, and urban planning.

We provide intuition as to why a vehicular network consisting of taxis is well suited to the problem of describing traffic patterns: we demonstrate empirically and theoretically that taxis, no matter what their initial locations are, tend to rapidly "spread out", thus providing good and consistent "coverage" of the road network, by showing that they behave like rapidly mixing Markov chains.

[1] Loop detectors are inductive loops installed in the road network, typically at intersections. They can detect metal and thus and count vehicles.

This work builds on prior studies on traffic to estimate traffic volume and speed [38] and mobility to measure the origins, destinations, and trajectories of trips [87].

1.5 Congestion Detection

Multi-agent congestion-aware algorithms require a congestion model that captures the relationship between flow and travel time. Learning this relationship is challenging. In this thesis we provide a method for learning the flow-travel time relationship, develop a congestion-aware routing system that uses it, and evaluate the multi-agent socially optimal path planning algorithm against real data. Specifically, we develop the congestion model using one month of data from 16,000 taxis and 10,000 loop detectors in Singapore.

In this thesis, we propose using loop detectors together with a new source of data—a roving fleet of dynamic sensor "probes"—to estimate in real time (1) the fundamental diagrams of road segments, which define their effective capacities, (2) the critical points in these fundamental diagrams, which effectively define the onset of congestion, and (3) the current operating level of the road. The critical congestion point of a road segment is defined as the road traffic state where the road capacity is reached and congestion starts. The ability to estimate critical congestion points off-line, as well as the ability to determine the current operating condition of a road segment relative to these critical points in real time, has the potential to make an enormous impact on urban planning. Knowing when a road state approaches congestion can be used by urban planners and road network users to make many decisions such as suggesting or taking alternative routes, setting congestion pricing schemes, and selecting public transportation routes and schedules.

Detecting congestion in real time is difficult, given standard techniques and data sources. However, GPS technology enables the estimation of the speed and position of vehicles. We propose a congestion estimation methodology based on estimating the fundamental relationship between traffic flow, density, and speed for any road segment off-line and then determining a road segment's state using this learned relationship and dynamic sensor data on-line. We demonstrate how all this information can be gleaned from taxi probes equipped with GPS sensors and existing loop detector data. Using the extensive data described in

Section 1.4, we develop a method for estimating the critical congestion point of a given road segment by (1) estimating the fundamental diagram for that road segment off-line and (2) estimating where in the fundamental diagram the current road state lies on-line.

In this thesis, we provide an efficient method for estimating road flow information for any road segment using the taxi sensor network and existing loop detectors. Since not all roads have loop detectors, we show that if we can estimate flow well, we can also accurately estimate the critical congestion point for the road. We propose a method that estimates the flow of a location without loop detectors using the real-time loop count information from "similar" locations and the real-time taxi count for the given location. We use the dynamic taxi sensor network, calibrated with static loops, to statistically infer fundamental diagrams and critical congestion points and dynamically evaluate road performance in relation to these critical points. We evaluate the accuracy of our method using the Singapore taxi and loop detector data.

We also discuss how the theoretical fundamental diagrams are an idealized model of the relationship between flow, density and speed. Specifically, the theoretical fundamental diagrams postulate two distinct operational states of 'free flow' and 'congestion.' In reality, we find that additional operational states, such as 'heavy traffic', may exist between the 'free flow' and 'congestion' states. Furthermore, the state of 'true congestion', where it is impossible to increase throughput, may never be empirically observed on a road. We show how the multiple operational states may be discovered from empirical data and distinguished from the 'true congestion' state.

1.6 System, Applications, and Experiments

Using the flow-delay function for each road segment, we build the congestion-aware route planning system with street-level congestion models. First, we estimate the flow of each road segment using the loop detector data and the taxi data. Next, we estimate the flow-delay function. We then implement the single-user congestion-aware algorithm [83] and the multi-agent path planning algorithm [84] to capture the dual types of optimization we wish to support in such a system: (1) the multi-agent social optimum and (2) greedy optimal planning. Using the system and taxi travel paths we show that the multi-agent social optimum algorithm improves the system performance by 15%.

1.7 Contributions

The main contributions of this thesis are as follows:

1. *Efficient exact stochastic motion planning algorithms*–A stochastic motion planning problem is presented which is general enough to represent many scenarios involving multiple agents, multiple resources, and multiple destinations. An efficient algorithm is developed to find an optimal agent, resource, locations to visit, and path coping with the travel time uncertainty.

2. *Approximation Algorithm for Stochastic Motion Planning by Pre-computation*–A speed-up method for stochastic shortest-path queries is developed with guaranteed approximation bound in optimality. The method is efficient both in query time and storage space.

3. *Congestion-aware Multi-agent Routing Algorithm*–We develop a multi-agent route planning algorithm and prove the convergence and rate of convergence. The algorithm is decentralized, using only local information; thus, it scales well for large network problems.

4. *Flow Estimation and Congestion Detection*–Data-driven methods of flow estimation and congestion detection are proposed and validated using real data from GPS sensors and inductive loop detectors.

5. *Implementation of Routing System*–Our traffic routing system uses the flow-delay function to estimate street-level congestion models and implements multi-agent congestion-aware routing with the social optimum guarantee.

6. *City-Scale Experiment*–We run experiments that show comparison between actual taxi paths, socially optimal congestion-aware routing, and greedy path planning.

1.8 Organization

This thesis is organized as follows: Chapter 2 summarizes related work. Chapters 3, 4, 5, and 6 present the traffic routing algorithms and their theoretical analyses. More specifically, Chapter 3 provides a single-origin single-destination stochastic route planning algorithm. Chapter 4 shows how we can speed up the stochastic shortest-path calculation using pre-computation. Chapter 5 presents stochastic routing in the presence of multiple agents and multiple visit locations. Chapter 6 gives a multi-agent routing algorithm considering the changing travel times in road segments depending on the agents' path choices. Chapters 7, 8, 9, and 10 provide the implementation of the theories and algorithms and experimental results. Chapter 7 describes our traffic routing system implementing the developed algorithms and the data used for the system. Chapters 8 and 9 present data-driven model estimation methods. Chapter 8 presents a method for traffic flow estimation using sensor data. In Chapter 9 we describe how we detect traffic congestion. Chapter 10 presents the experimental results of our traffic routing system using real data, estimated congestion models, and our algorithms. Chapter 11 summarizes this thesis, provides lessons learned, and outlines possible future research.

Chapter 2

Related Work

Our proposed algorithms and systems are further related to prior work in the fields of stochastic shortest paths, distance oracles for planar graphs, multi-agent routing, traffic prediction and congestion estimation. In this chapter, we summarize related works.

2.1 Stochastic Routing

Our work is closely related to stochastic planning and stochastic shortest-path algorithms. Prior work has considered stochastic shortest paths under the assumption that the travel time follows a known probability distribution [118, 21, 88, 129, 96, 101, 102]. In stochastic shortest path, edge costs are probability distributions rather than deterministic costs and the optimal path depends on drivers' diverse objectives. When a driver's objective is to minimize the expected travel time, the problem becomes the deterministic shortest-path problem. It is well known that Dijkstra's algorithm is optimal and applicable for deterministic problems. However, for some 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.

Stochastic shortest-path problems are characterized by stochastic instead of deterministic edge weights [21, 88, 96, 118, 129, 101, 102, 100, 83]. Because the edge weight is represented as a probability distribution rather than a fixed value, there can be many different optimal path definitions. A few frequently used optimality criteria are [83]:

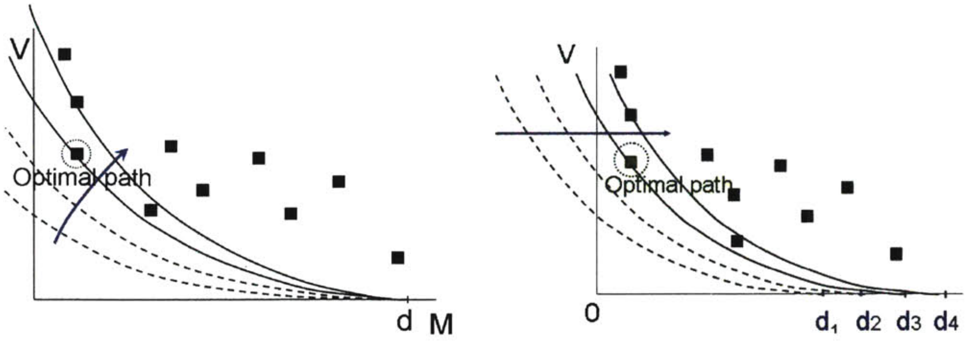


Figure 2-1: Visualization of Stochastic Shortest Paths. Stochastic shortest paths optimize drivers' goals considering the probability distribution of delays for road segments. **Left:** The goal is finding a path that maximizes the probability of reaching the destination within a given deadline *d* (Probability-Tail Model). **Right:** The goal is finding a path that allows the latest departure, guaranteeing that the probability of making the deadline is at least a given probability value (Mean-Risk Model). X axis and Y axis of the plots represent the mean and variance of the travel time statistics for a path. Reprinted from [83].

- Given an origin and a destination, find a path that maximizes the probability of reaching the destination within a given deadline. (Probability-Tail Model)
- Given an origin and a destination, find a path that allows the latest departure guaranteeing that the probability of making the deadline is at least a given probability value. (Mean-Risk Model)

There have been many studies on single-origin single-destination stochastic shortest-path problems. Nikolova et al. [102] developed an algorithm and theoretical bounds by 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 [83]. In this thesis, we focus on stochastic shortest-path problems with multiple origins, multiple destinations, and multiple intermediate visit node constraints. This thesis builds on the single-origin single-destination stochastic shortest-path algorithm developed in [83]. We show how to utilize the parametric optimization block in [83] to solve the constrained stochastic shortest-path problem. Related prior research considers the visit node constraint problems in deterministic graphs [33, 47, 68].

There is some research on the shortest-path problem with visit node constraints [137, 33] in a deterministic setting. Significant research has been done on the stochastic shortest-path problem [109, 95, 28, 45, 107, 118, 97, 82, 83, 88, 96, 101, 102]. [23] developed a smartphone based routing system that maximizes the probability of reaching the destination within a given time budget. However, most of the stochastic shortest-path research has focused on the single-origin single-destination case without any constraints. We extend the prior work by providing a solution to the stochastic shortest-path problem for multiple agents with visit node constraints on a stochastic network.

2.2 Approximate Distance Oracles

An (approximate) distance oracle is a data structure that efficiently answers shortest-path and/or distance queries in graphs. Relevant quantities of a distance oracle include the *preprocessing time*, which is the time required to construct the data structure, the *space* consumption, the *query time*, and, for approximate distance oracles, the *stretch*, which is the worst-case ratio among all pairs of query nodes of the query result divided by the actual shortest-path distance. Approximate distance oracles using quasilinear space for poly-logarithmic query time and stretch $(1+\varepsilon)$ are known for planar [123] (implementation in [94]), bounded-genus [70], minor-free [11], geometric [57], and bounded-doubling-dimension graphs [16]. For realistic traffic scenarios, we need the distance oracle to work with directed graphs (also called *digraphs*). For planar digraphs, Thorup [123] gave an approximate distance oracle that occupies space $O(n\varepsilon^{-1}\log(nL)\log(n))$ and answers $(1+\varepsilon)$–approximate distance queries in time $O(\varepsilon^{-1}+\log\log(nL))$, where L denotes the largest integer length of edge weights (see [70] for different tradeoffs between space and query time).

Furthermore, many efficient practical methods have been devised [52, 115, 17], their time and space complexities are however difficult to analyze. Competitive worst-case bounds have been achieved under the assumption that actual road networks have small *highway dimension* [10, 9].

2.3 Multi-Agent Routing

There have been many studies in finding the social optimum and the user equilibrium in traffic [77, 114, 31, 69]. Incremental planning for multiple drivers was studied in [131] using the assumption that the stochastic travel time for a road segment changes according to the number of previously planned drivers. Our work is closely related to traffic control problems of multiple agent systems, multicommodity-flow problems, and congestion game problems [44, 80, 128, 78, 32, 117]. The multi-commodity flow problem is a network flow problem with multiple commodities (or goods) flowing through the network, with different source and sink nodes. There have been many studies on congestion control in communication networks [72, 71, 89, 12]. In game theory, many studies have been done to find user equilibrium [127, 49, 50, 19]. There are studies in fractional flow settings [40, 14, 105]. In this thesis, we treat the probability of agent's path choice as fractional flow. Our research finds the path choice probability of each agent to achieve the social optimum. The algorithm for finding the probability is deterministic, but the actual assignment of agents to a path is stochastic according to the probability. Our probability of path can be viewed as similar to the mixed strategy in game theory. In [110], the convergence result was given for the previous algorithm called the method of successive averages. The research provides the convergence result, but the rate of convergence is not given. Our work provides the convergence speed. In addition, [110] requires the global information. Our work only requires local information, so the algorithm can run in a parallelized way.

In [36], stochastic user equilibrium (SUE) was introduced. The probability of route choice is modeled as a logit distribution according to the measure travel times of K paths for each driver. We also use this concept and find the optimal probability of route choice. The law of large numbers is the basis for the construction of link flows. The link flow is the summation of all the path probability for all the paths that have the link in the path. The stochasticity explains the route choice probability of individuals due to the perception errors. The stochasticity is about the random choice based on perception error and is not about the stochastic travel time. In this work and following many studies, the probability of path choice is considered as fractional flow for the path. [75] developed an algorithm for finding stochastic user equilibrium solution for congested transit networks.

[119] compares the implementation of different traffic assignment methods.

2.4 Traffic Data

The state of the art in estimating traffic uses static sensors such as loop detectors or traffic cameras [6]. The static sensors are installed at fixed points and provide traffic estimates at that single location. They require significant effort to deploy and maintain, and they fail to capture traffic flow and trajectory information. Some recent studies have began to investigate the use of GPS devices as dynamic traffic probes for inferring traffic volume using existing mathematical models [55, 62], or they estimate traffic speed from GPS [134, 133] or from special-purpose static sensors [3, 4].

Measuring mobility patterns is more challenging than measuring traffic volume. The state-of-the-art methodology for recording the origins and destinations (OD) of trips is a manually-conducted survey [39, 113, 7, 8]. OD surveys include household, workplace, or roadside surveys and aim to identify where people start and end their trips. There are many shortcomings with this method. The surveys measure the average rather than actual travel behavior, they cover only a small subset of trips, and given the manual nature of the method, the information (e.g. travel time) is poorly estimated by the interviewee. More recently, estimating origin and destination has also been done using the observed link flow information [27, 58, 22, 13, 61, 59, 60]. However, the results of this method depend on an underlying model of traffic and the number of link flow measurement locations. Surveys have also been used to estimate route choices, although the results are not reliable due to the complexity and scale of the route selection problem in a dense network of roads. Truck fleets have been used to identify truck routes using loop detector counts [112]. Table 2.1 summarizes these previous studies according to the data sources used and the information extracted from them.

Table 2.1: Summary of data sources

Data Info	Survey	Static sensors	Dynamic sensors
Origin and destination	Household survey, workplace survey, Roadside interview [39, 113, 7, 8]	OD estimation technique with observed link flows [27, 58, 22, 13, 61, 59, 60]	
Route		Reconstruct truck routes from loop detector counts [112]	
Flow		Loop detectors and cameras, etc. [6]	Estimate flow from GPS data [62]
Speed		Advanced loop detectors and cameras, etc.	From GPS[134, 133]

2.5 Traffic Prediction and Congestion Modeling

Many traffic routing systems have been developed based on travel time estimation from GPS data collected by probe vehicles [65, 138, 37, 48, 82, 63].

[65] developed a sensor network system that collects cars' traces while they are moving and transmits the GPS information through open Wi-Fi access points. For probe vehicles to be the basis of a real-time traffic monitoring system, it is necessary that the travel-time measurements made by them be reliable, adequate and timely. In [66], coverage of roads for a traffic congestion monitoring application is provided based on simulations. [138] proposed a method of identifying traffic conditions and patterns on surface streets given location traces collected from on-road vehicles and infrequent low-bandwidth cellular updates. [37, 48] estimated the number of probe vehicles required for various types of road networks using simulation. [135] uses machine learning techniques to predict the travel time.

Prior research for estimating traffic flow includes [34, 122, 121]. [73] uses simulation tools to address macroscopic simulation of a real large-scale motorway network. [42] uses statistical learning theory to predict flow. [64] predicts traffic flow and congestion for a small set of known bottleneck regions by using a large set of historical and realtime data to construct a Bayesian network. However, the algorithm in [64] cannot predict the traffic on every roadway and street in a city, nor can it predict the fastest route between two arbitrary points in a city. [111] uses neural network methods to estimate traffic flow.

The fundamental diagram of a road describes the relationship between flow, density, and speed on a road [106, 29, 35, 46, 41, 74, 15, 76, 136, 139, 140].

Chapter 3

Single-Agent Route Planning Under Uncertainty

This chapter presents a single-origin single-destination stochastic shortest path algorithm. The algorithms in Chapter 4 and Chapter 5 build on the algorithm described in this chapter.

Section 3.1 provides the set-up for the stochastic network, and Section 3.2 describes the stochastic traffic model using GPS traces. Section 3.3 summarizes the stochastic shortest-path algorithm in [83].

3.1 Stochastic Network

We model the road network as a directed graph $G = (V, E)$ consisting of a set of vertices V that represent road intersections, and edges $E \subset V \times V$ that represent road segments between intersections. Associated with each edge e is the travel cost distribution $w_e \sim \mathcal{N}(m_e, v_e)$, which is assumed to be a normal distribution with mean m_e and variance v_e of the travel cost of e. We assume that this distribution is independent from travel cost distributions at other road segments.[1]

[1] Stochastic dependency between adjacent edges can be considered by transforming the given graph as shown in Section 5.3.2

3.2 Stochastic Traffic Model based on GPS Traces

We compute a stochastic model of an entire city using the following process. We start with a heterogeneous set of roughly 500 million anonymized GPS points. Grouped by individual vehicles, we extract those traces (subpaths), for which the sampling frequency is sufficiently high (at least one data point every thirty seconds). For each trace, we employ map-matching algorithms to compute the sequence of road segments that has the largest likelihood to have been used by this particular trace. The process of matching GPS trace data into a map has been investigated before [130, 56]. To overcome the noise and sparsity of GPS data, a map matching method based on the Viterbi algorithm was suggested in [99]. We use their method for our dataset. Since we do not have the ground truth for our dataset, the correctness of the map-matching step cannot be verified. It is actually rather unlikely that all traces are mapped to the correct sequence of road segments. We assume that our dataset is large enough such that the majority of traces is mapped to correct road segments. We determine the speed of each vehicle per road segment and we sort all the values for each road segment by time. We then bin speed values into one-hour intervals for each day of the week. Using standard statistics, we compute the sample mean and standard deviation for the speed and travel time for each bin.

Figure 3-1 shows distributions for different road segments and different times of the day.

3.3 Efficient Solution for Single Agent Stochastic Shortest Path (SSP) Problem

This section summarizes the solution method for SSP given in [83] on which this chapter builds. [83] presents a method that finds the optimal path when the optimality criterion is either of those given in Section 2.1. Let path π be denoted by a point (m_π, v_π) in a rectangular coordinate system, called the mean-variance plane, where the horizontal axis represents the mean and the vertical axis represents the variance. As explained in detail in Section 3.3.2, level sets of objective functions are shaped in parabola. Figure 3-2(a) shows

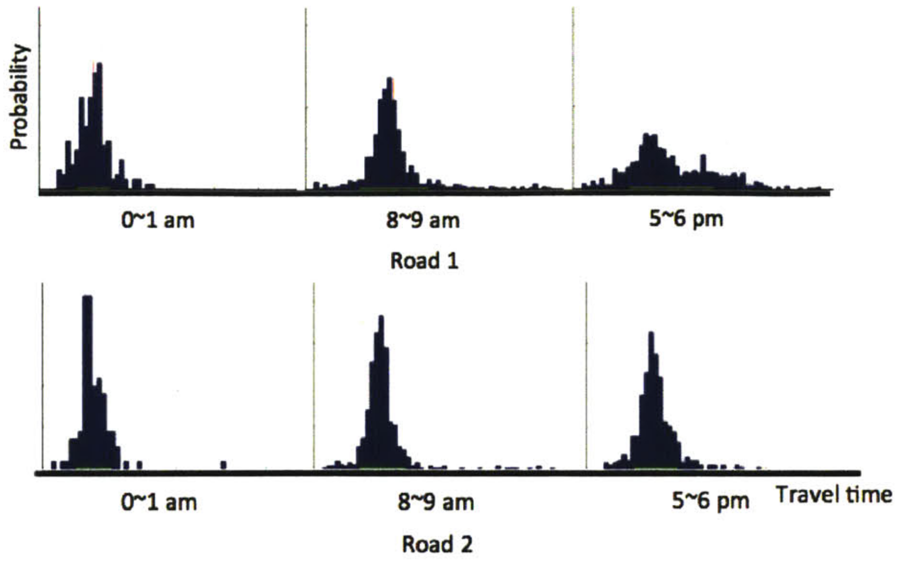


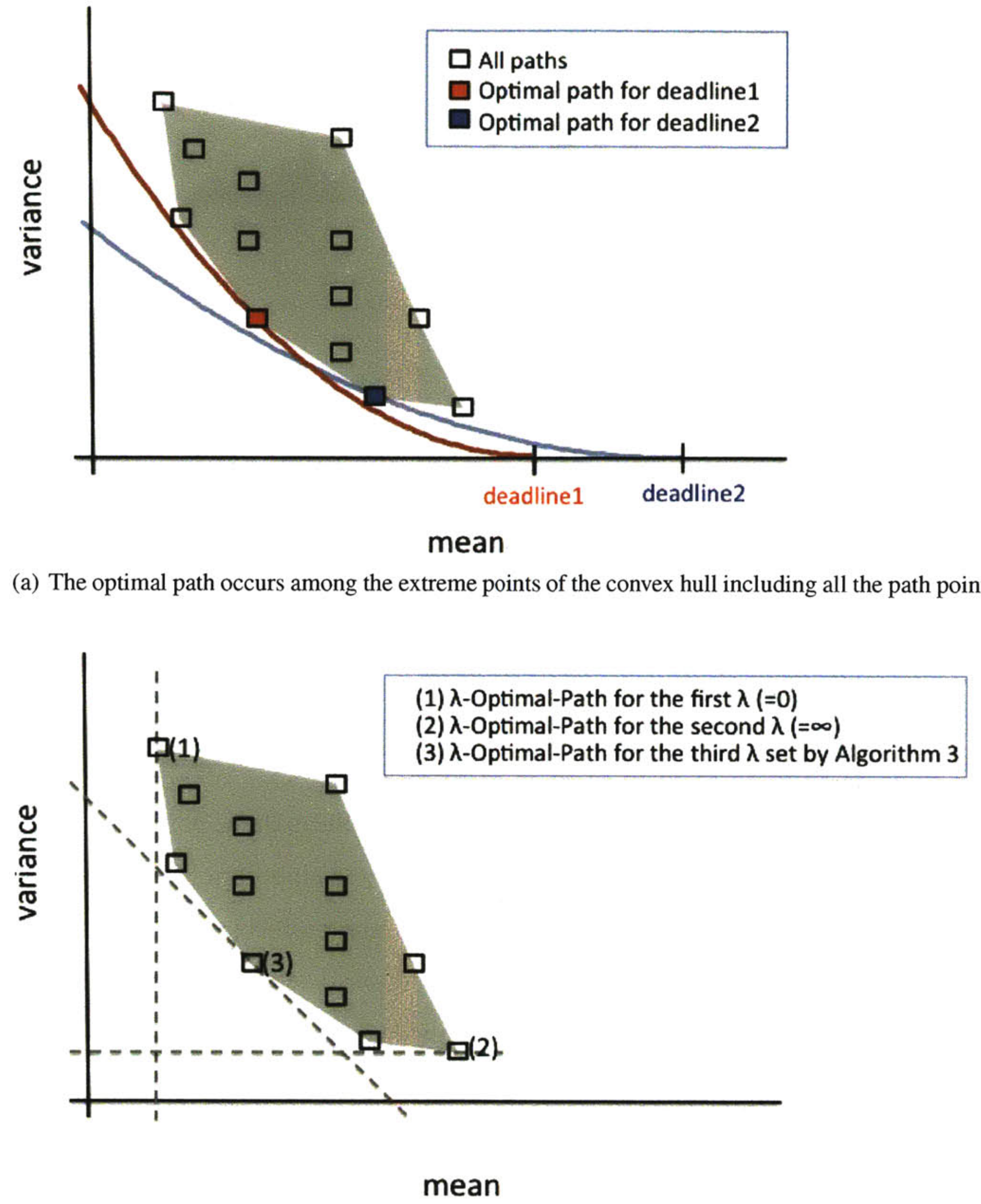
Figure 3-1: The travel time distribution of two different road segments for 0∼1 am, 8∼9 am, and 5∼6 pm on weekdays.

two different maximum arrival probability paths depending on the deadlines.

The key property is that the optimal path occurs among the extreme points of the convex hull containing all the path points. Using this property, the algorithm solves the problem by finding the λ*-optimal path*, which is defined as the minimum-cost path where the edge weight is given by $w_e(\lambda) = m_e + \lambda \cdot v_e$ for any nonnegative λ. We refer to this linear combination of mean and variance as λ*-cost*. From the independency assumption between edge distributions, it follows that a path's λ-cost is the sum of the λ-costs of all the edges in the path. Thus, the λ-optimal path can be found using Dijkstra's algorithm or any other standard shortest-path algorithms. The algorithm is shown in Algorithm 1, and a visualization of the iterations is given in Figure 3-2(b).

Algorithm 1 finds and prunes the search regions and selects only a small number of λ values. The algorithm's building blocks (Algorithms 2, 3, and 4) are used in the algorithms for visit node constraints in Sections 5.1 and 5.2. The **isBetterThan** operator in Algorithm 4 compares the two paths depending on the optimality criterion. Deadline information is used for this comparison.

The details of the SSP algorithm in [83] are further described in Sections 3.3.1 and 3.3.2.



(a) The optimal path occurs among the extreme points of the convex hull including all the path points.

(b) The extreme points can be found efficiently by finding λ-optimal paths where λ values are selected by Algorithm 3.

Figure 3-2: Explanation of the SSP algorithm in [83]

3.3.1 Problem Formulation

Road Network Modeling

The road network is a graph called the Geographic Map, where nodes represent intersections and edges represent road segments. We associate a road delay distribution with each

Algorithm 1: Find-Optimal-Path

```
Data: Graph, origin and destination, deadline
Result: optimal path from origin to destination
optimalPath := NULL;
while ($\lambda$:=Find-Next-$\lambda$())$\neq$NULL do
    $\pi$ := Find-$\lambda$-Optimal-Path($\lambda$, origin, destination);
    optimalPath := Update-Regions-and-Optimal-Path($\pi$, deadline);
end
return optimalPath
```

Algorithm 2: Find-λ-Optimal-Path

```
Data: $\lambda$, start, end
Result: $\lambda$-optimal path from start to end
Find the path minimizing $mean + \lambda \cdot variance$ using a shortest-path algorithm such
as Dijkstra's or A* search algrotihm;
```

Algorithm 3: Find-Next-λ

```
Data:
Result: $\lambda$ value for the candidate region to be searched if there exists a region, or
        NULL otherwise
$\lambda$ := NULL;
while (R :=pop from the candidate region FIFO queue)$\neq$NULL do
    if the region satisfies the condition of pruning then
        continue;
    end
    else
        return $\lambda$ := lambda value for the region
    end
end
return $\lambda$
```

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 GPS traces onto real road segments, while the Delay Statistics Map is used for statistical delay sensitive routing. We assume that

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

Algorithm 4: Update-Regions-and-Optimal-Path.

```
Data: new $\lambda$-optimal path $\pi$, deadline
Result: Updated optimal path and candidate regions
if $\pi$ isBetterThan currentOptimalPath then
    currentOptimalPath = $\pi$;
end
Calculate new two candidate regions;
for each candidate region do
    if the probe point of candidate region isBetterThan currentOptimalPath then
        Insert the region into the candidate regions list;
    end
end
return currentOptimalPath
```

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:

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

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 $C(t) = e^{kt}$, where k is the steepness of the cost increase, the expected cost can be written as $EC_\pi = \int_{-\infty}^{\infty} e^{kt} f_\pi(t)dt = \{e^{k(m_\pi+\frac{kv_\pi}{2})}\}$. This exponential cost function minimizes a linear combination of mean and variance determined by k.

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 $EC_\pi(d) = \int_{-\infty}^{\infty} u(t-d) f_\pi(t)dt = \int_d^\infty f_\pi(t)dt = \{1-\Phi(\frac{d-m_\pi}{\sqrt{v_\pi}})\}$, 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 $\text{argmax}_{\pi\in\Pi}\ \Phi(\frac{d-m_\pi}{\sqrt{v_\pi}})$, which turns out to be the path that maximizes arrival probability. Since $\Phi(\cdot)$ is monotonically increasing, maximizing $\Phi(\cdot)$ is equivalent to maximizing

$$\varphi_d(\pi) = \frac{d-m_\pi}{\sqrt{v_\pi}}. \tag{3.1}$$

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 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, (3.1), is nonlinear. Our goal for the rest of this chapter is to develop an efficient algorithm for finding the maximum probability path given a deadline.

3.3.2 Stochastic Path Planning by Parametric Optimization

In [102], 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 a parametric optimization problem, which will ultimately lead to a new algorithm that reduces unnecessary runs over [102].

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 mean-variance plane, where the horizontal axis represents the mean and the vertical axis represents the variance. The objective of the optimization problem, (3.1), can be rewritten to show the relation between m_π and v_π as

$$v_\pi = \frac{1}{\varphi_d(\pi)^2}(m_\pi - d)^2, \tag{3.2}$$

which is a parabola in the mean-variance 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. 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 3-3 (Left)). This suggests finding the optimal path using linear optimization with various combinations of cost coefficients.[2]

Consider setting the cost of an edge to 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 [102].

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

Proof. Let point P on the mean-variance 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 algorithm) will find λ-optimal paths.

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

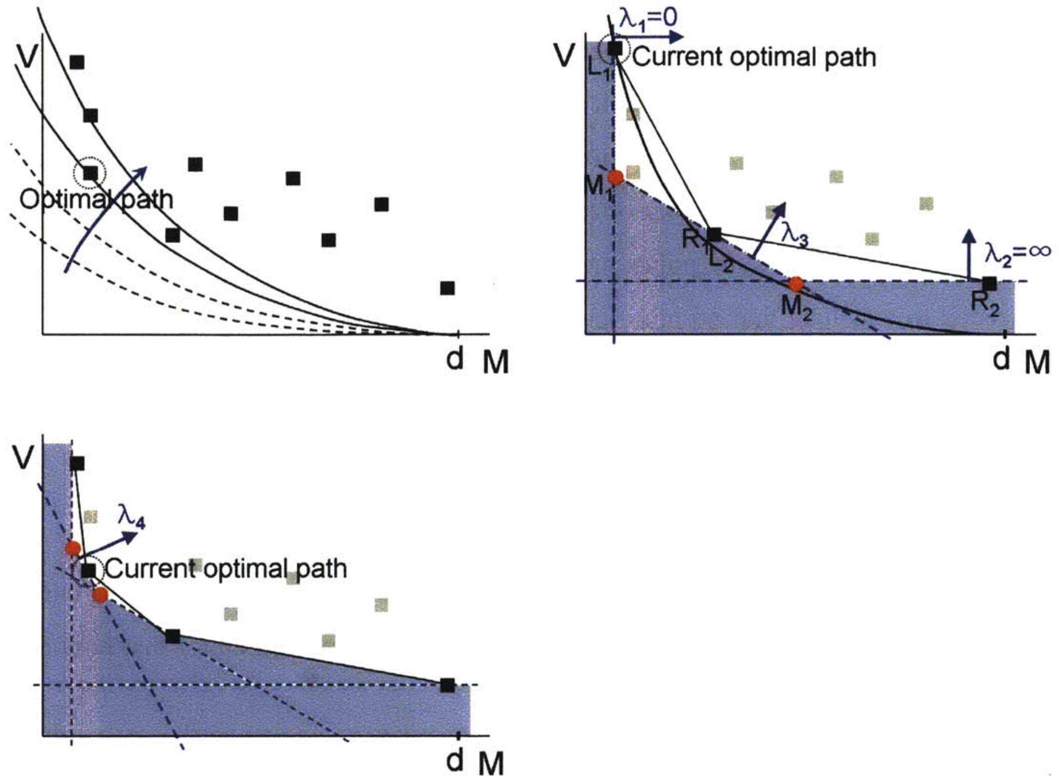


Figure 3-3: (Top left) Graphical interpretation of the optimal path in the mean-variance plane. Each square represents a path from the origin to the destination. Equi-probability paths lie on a parabola with an apex at $(d,0)$ and a curvature of $\frac{1}{\varphi_d(\pi)^2}$. The optimal path is the first point that meets with a parabola as we increase the curvature. (Top right) 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 square 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 round points are the probe points of the regions. (Bottom left) λ-optimal search was done only for the left candidate region. The newly found path turns out to be the new current optimal path and the two round points are the probe points.

Exhaustive Enumeration

In [102], 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 3-3 (Middle). 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 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.

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 $\triangle L_iM_iR_i$, and probes as round points M_i in Figure 3-3 (Middle).*

Theorem 1. *If the φ value of the probe point as defined in (3.1) 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 3-3 (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 the current optimal path. The left candidate region $\triangle L_1 M_1 R_1$ was searched since the left probe point gives a 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.

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

$$\begin{aligned} \lambda_u(v_{\lambda_u} - v_\lambda) &< m_\lambda - m_{\lambda_u}, \\ \lambda(v_{\lambda_u} - v_\lambda) &> m_\lambda - m_{\lambda_u}. \end{aligned} \tag{3.3}$$

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 derivitive of (3.2) and its negative inverse $\lambda_u = -1/\frac{\partial v}{\partial m}|_{m=m_0} = \frac{(d-m_0)^2}{2v_\infty(d-m_0)} = \frac{d-m_0}{2v_\infty}$. From this and (3.3), and since $d - m_0 > d - m_\lambda$ and $v_\lambda > v_\infty$,

$$\frac{m_\lambda - m_{\lambda_u}}{v_{\lambda_u} - v_\lambda} > \frac{d - m_0}{2v_\infty} > \frac{1}{2}\frac{d - m_\lambda}{v_\lambda}. \tag{3.4}$$

Since $m_\lambda > m_{\lambda_u}$ and $d - m_\lambda > 0$, $\frac{1}{\frac{m_\lambda - m_{\lambda_u}}{d - m_\lambda} + 2} < \frac{1}{2}$. From this and (3.4),

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

We can rewrite this as follows:

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

Thus, for any $\lambda > \lambda_u$, the λ-optimal solution is worse than the 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 $\lambda \leq \lambda_u$. □

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

$$\lambda_l(v_{\lambda_l} - v_\lambda) < m_\lambda - m_{\lambda_l}, \tag{3.5}$$

$$\lambda(v_{\lambda_l} - v_\lambda) > m_\lambda - m_{\lambda_l}. \tag{3.6}$$

Inequalities (3.5) and (3.6) 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 (3.2) and its negative inverse

$$\lambda_l = -1/\frac{\partial v}{\partial m}|_{m=m_0} = \frac{(d-m_{opt})^2}{2v_{opt}(d-m_0)} \tag{3.7}$$

From (3.5) and (3.7),

$$\frac{m_\lambda - m_{\lambda_l}}{v_{\lambda_l} - v_\lambda} < \frac{\varphi_d(\pi_{opt})^2}{2(d-m_0)}. \tag{3.8}$$

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

$$\frac{\varphi_d(\pi_{opt})^2}{2(d-m_0)} \leq \frac{\varphi_d(\pi_\lambda)^2}{2(d-m_0)} < \frac{\varphi_d(\pi_\lambda)^2}{2(d-m_\lambda)} = \frac{d-m_\lambda}{2v_\lambda} \tag{3.9}$$

since $d-m_0 > d-m_\lambda$. From (3.8) and (3.9)

$$\frac{m_\lambda - m_{\lambda_l}}{v_{\lambda_l} - v_\lambda} < \frac{1}{2}\frac{d-m_\lambda}{v_\lambda} \tag{3.10}$$

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

$$\frac{1}{\frac{m_\lambda - m_{\lambda_l}}{d-m_\lambda}+2} > \frac{1}{2} \tag{3.11}$$

From equation (3.10) and equation (3.11),

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

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. □

Theorems 1, 2, and 3 lead to the Parametric Search algorithm (see Algorithm 5) for finding the best route that maximizes the probability of arriving at the destination within a given deadline. In lines 3 and 4, the 0-optimal and ∞-optimal paths are searched with a shortest-path algorithm (e.g., Dijkstra's algorithm or A^* search algorithm). 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 into the queue. Candidate regions

Algorithm 5: PARAMETRIC-SEARCH

```
Data: Graph with mean and variance of each edge, origin, and destination
Result: The optimal path
 1: $bestPath \leftarrow \varnothing$
 2: $Regions = []$ : 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: $Regions.push(Region(l : path_0,\ r : path_\infty,$
      $p : (path_0.mean, path_\infty.var)))$
 8: calculate $\lambda_l$ and $\lambda_u$
 9: while $(R \leftarrow Regions.pop())\ != \varnothing$ 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 $path\ != R.l$ and $path\ != R.r$ then
20:         if $path.\varphi > bestPath.\varphi$ then
21:             $bestPath \leftarrow path$, update $\lambda_l$
22:         locate $probe_l$ and $probe_r$
23:         if $probe_l.\varphi > bestPath.\varphi$ then
24:             $Regions.push(Region(l : R.l,\ r : path,$
                  $p : probe_l))$
25:         if $probe_r.\varphi > bestPath.\varphi$ then
26:             $Regions.push(Region(l : path,\ r : R.r,$
                  $p : probe_r))$
27: return $bestPath$
```

are evaluated for searching. The conditions in lines 10, 23, and 25 come from Theorem 1, and those in lines 12 and 15 from Theorems 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 (lines 13 and 16) in line 18.

Correctness

Algorithm 5 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 3.3.2, the algorithm finds the optimal solution in a finite number of searches.

Running Time

As shown by Nikolova, et al. in [102] based on [26], there are total $N^{\Theta(logN)}$ extreme points in the mean-variance plane, where N is the number of nodes in the network. Compared to their algorithm that searches every extreme point, our algorithm does not invoke unnecessary searches yielding an average running time of $O(N^2log^4N)$, where N^2 term is due to Dijkstra's runs for each λ-optimal search.

The intuition is as follows. Each new path point found with a λ-optimal search yields two candidate regions. Our algorithm only searches the candidate region if its probe point is outside the current optimal parabola. The parabola passing through the newly found path point will almost always divide the two probe points causing one of them to lie outside the parabola. The adverse case where both probe points are outside the parabola happens rarely when the current optimal parabola meets the current λ-optimal search line twice within the interval determined by the two probe points. The decision to remove candidate regions without searching them depends on the distribution of path points in the mean-variance plane and the deadline. Thus, the running time of our algorithm can be described probabilistically.

We let $p_2(i)$ be the probability that both candidate regions are searched at the i'th iteration and $p_0(i)$ be the probability that neither candidate region is searched at the i'th iteration. We give the running time bound of our algorithm taking the varying $p_2(i)$ and $p_0(i)$ into account in Theorem 4. We use an assumption of how $p_2(i)$ and $p_0(i)$ vary as algorithm iterations proceed (Assumption 1) and a lemma for the running time of our algorithm when $p_2(i)$ and $p_0(i)$ satisfy a certain condition (Lemma 2).

To see how $p_2(i)$ and $p_0(i)$ grow as iterations proceed, we observed them by running our algorithm on randomly generated convex hulls. We generated random convex hulls with a fixed number of extreme points, ran our algorithm for each instance of the convex hull, and observed the occurrence of the event at each iteration. The event is one of 'N', 'O', and 'B', meaning none, one, and both of the candidate regions are searched, respectively. After observing these for each instance of convex hulls, we find the empirical probabilities as follows:

$$p_2(i) = \frac{\text{The number of 'B' at } i\text{'th iteration}}{\text{The number of 'N', 'O', and 'B' at } i\text{'th iteration}}.$$

$$p_0(i) = \frac{\text{The number of 'N' at } i\text{'th iteration}}{\text{The number of 'N', 'O', and 'B' at } i\text{'th iteration}}$$

Figure 3-4 shows the plots of $p_2(i)$ and $p_0(i)$ where 5000 random convex hulls were used. As denoted at the top of each plot in Figure 3-4, $p_2(i)$ becomes smaller than $p_0(i)$ at around the $logN_e$'th iteration, where N_e is the number of extreme points and remains smaller for the following iterations.

Remark 1. *The reasoning behind this empirical data is as follows: First, consider how $p_2(i)$ and $p_0(i)$ vary as iterations proceed. We have three cases: (a) two regions are left in the queue, (b) one region is left in the queue, or (c) no region is left in the queue. Case (a) occurs only when the λ-optimal search line is in the neighborhood of the tangent to the parabola at the point being considered. The size of the neighborhood depends on the distance between the two probe points (e.g., if the distance is large, even a large difference in the slopes can result in the retention of both points.) If the distance is small, only a*

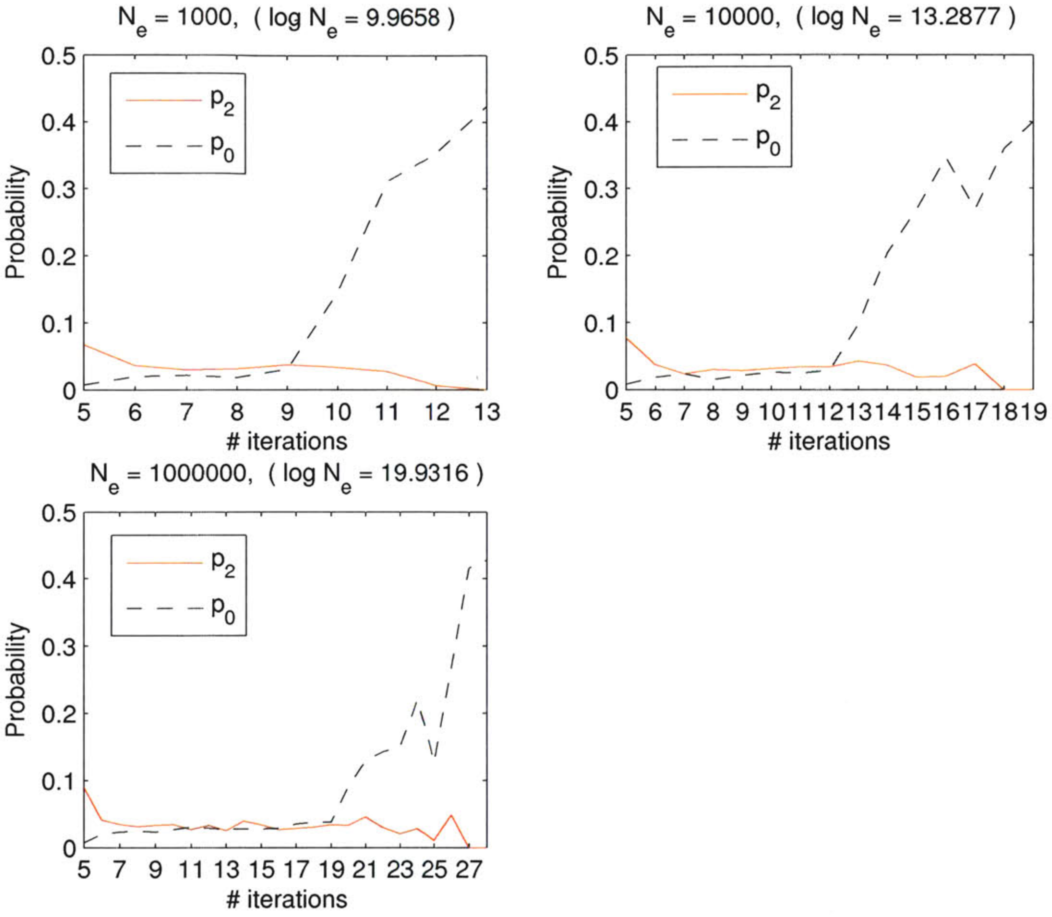


Figure 3-4: p_2 and p_0 was calculated from 5000 random convex hulls with N_e extreme points.

small difference in the slopes will lead to the retention of both points. Since the distance gets smaller as iterations proceed, $p_2(i)$ decreases monotonically. On the other hand, the probability of case (c) (e.g., both candidate regions are removed) monotonically increases as the distance between the probe points gets smaller. The optimal parabola gets flatter and the lower bound of λ (λ_l) increases as a better λ-optimal path is found, which also causes $p_2(i)$ to decrease and $p_0(i)$ to increase.

Consider the point when $p_2(i)$ becomes smaller than $p_0(i)$. When a single child is retained at each step, it takes $O(logN_e) = O(log^2N)$ iterations until the optimal path is found because we have a binary tree. Thus the probability that the search algorithm retains one child decreases fast beyond $O(log^2N)$ iterations. This results in $p_0(i)$'s fast increase *since its increasing rate is related to the decreasing rate of* $1 - p_2(i) - p_0(i)$ *by* $\frac{\Delta p_0(i)}{\Delta i} = -\frac{\Delta(1-p_2(i)-p_0(i))}{\Delta i} + (-\frac{\Delta p_2(i)}{\Delta i})$. *Thus,* $p_0(i)$ *becomes larger than* $p_2(i)$ *around* $i = O(log^2N)$. *Figure 3-4 illustrates these behaviors of* $p_2(i)$ *and* $p_0(i)$.

Based on the empirical observation about $p_2(i)$ and $p_0(i)$ and Remark 1, we make the following assumption.

Assumption 1. $p_2(i) \leq p_0(i), \forall i > n, n = O(log^2N)$.

Lemma 2. *If* $p_2(i) \leq p_0(i), \forall i$, *the average running time of our algorithm is* $O(N^2log^2N)$, *where N is the number of nodes in the graph.*

Proof. We use a binary tree to represent a hierarchy of candidate regions created by the λ-search sequence. Each node represents a candidate region and its children nodes represent the two candidate regions created by searching the current region. Let h be the height of the tree, and let the function $num(\mathscr{N})$ be defined as the total number of candidate regions to search for a tree with a root node $\mathscr{N}$ and let $\mathscr{L}$ and $\mathscr{R}$ be the left and right children nodes of $\mathscr{N}$. Then, $h = O(log(N^{logN})) = O(log^2N)$ [102] and

$$num(\mathscr{N}) = \begin{cases} 1 + num(\mathscr{L}) + num(\mathscr{R}) & \text{with probability } p_2(i), \\ 1 + num(\mathscr{L}) & \text{with probability} \frac{1-p_2(i)-p_0(i)}{2}, \\ 1 + num(\mathscr{R}) & \text{with probability} \frac{1-p_2(i)-p_0(i)}{2}, \\ 1 & \text{with probability } p_0(i). \end{cases}$$

The conditional expectation of $num(\mathscr{N})$ given $num(\mathscr{L})$ and $num(\mathscr{R})$ is described as follows:

$$\begin{aligned} &E[num(\mathscr{N})|num(\mathscr{L}), num(\mathscr{R})] \\ &\quad = 1 + \frac{1 + p_2(i) - p_0(i)}{2}(num(\mathscr{L}) + num(\mathscr{R})) \end{aligned}$$

Then, the expectation of $num(\mathscr{N})$ can be expressed as follows, taking expectation over

$num(\mathscr{L})$ and $num(\mathscr{R})$ on the above conditional expectation.

$$\begin{aligned} E[num(\mathscr{N})] &= E[E[num(\mathscr{N})|num(\mathscr{L}),num(\mathscr{R})]] \\ &= 1+\frac{1+p_2(i)-p_0(i)}{2}\left(E[num(\mathscr{L})]+E[num(\mathscr{R})]\right) \end{aligned} \tag{3.12}$$

Since the expectation of $num(\mathscr{N})$ depends only on the height of the tree, we can define a function $num_e(h)$ as the expected number of nodes given the height h. Then, for a node $\mathscr{N}$ whose height is h and whose two children are $\mathscr{L}$ and $\mathscr{R}$, we have $E[num(\mathscr{N})] = num_e(h)$ and $E[num(\mathscr{L})] = E[num(\mathscr{R})] = num_e(h-1)$. From (3.12), $num_e(h)$ can be written as:

$$\begin{aligned} &num_e(h) \\ &= \begin{cases} 1+(1+p_2(i)-p_0(i))\ num_e(h-1) & \text{if } h \geq 1, \\ 0 & \text{if } h = 0. \end{cases} \end{aligned}$$

Solving the above recursive equations, noting that $1+p_2(i)-p_0(i) \leq 1$, we get:

$$num_e(h) \leq h = O(log^2 N).$$

Thus, the running time is $O(N^2 log^2 N)$ when we use Dijkstra's algorithm for each λ-optimal search.

□

Theorem 4. *The average running time of Algorithm 5 is $O(N^2 log^4 N)$.*

Proof. The total running time is determined by considering the computation before and after $p_2(i) \leq p_0(i)$ is satisfied separately. By Assumption 1 the computational cost before $p_2(i) \leq p_0(i)$ is satisfied is $O(N^2 log^2 N)$. Then, we have $O(log^2 N)$ candidate regions. In the worst case, all the remaining candidate regions should be searched. Since the height of the sub-trees spanning from those candidate regions is bounded by h and they all satisfy $p_2(i) \leq p_0(i)$, the computational cost after $p_2(i) \leq p_0(i)$ is satisfied is $O(N^2 log^4 N)$ by Lemma 2. Thus, the overall running time is $O(N^2 log^4 N)$. □

Stopping Earlier within Bounded Error

A sub-optimal path can be computed by stopping the search of the algorithm early. In such cases we can still guarantee an error bound. For example, we can compute faster solutions that are within 3% of the optimal solution.

When the probability difference between the current optimal path and the virtual path defined by the probe point is in the range of a user's tolerable error range, it is guaranteed that there is no other path that has larger probability at least by the tolerable error range. Thus, we can remove the candidate region from consideration. By doing this, more candidate regions can be removed from the queue, resulting in faster search. Here, the probe point can be used not only for comparing if the candidate region needs to be searched for a better solution, but also for deciding how much better the solution can be, if it exists. When the tolerance in probability is denoted as δ, line 10 of Algorithm 5 should be changed to

$$\Phi(R.probe.\varphi) - \Phi(bestPath.\varphi) < \delta,$$

and lines 23 and 25 of Algorithm 5 should be changed to

$$\Phi(probe_l.\varphi) - \Phi(bestPath.\varphi) > \delta$$

and

$$\Phi(probe_r.\varphi) - \Phi(bestPath.\varphi) > \delta,$$

respectively. Note that the algorithm with zero tolerance reduces to the original exact algorithm.

Extension to Latest Departure Time

We examined how to find the maximum probability path. The question was

- Given the *desired arrival time* and the *departure time*, what is the path that has the largest *probability of making the trip within a given deadline*?

Note that deadline is expressed as *desired arrival time - departure time.*

We can also think of another question,

- Given the *desired arrival time* and the *probability of making the trip within a given deadline*, what is the path that has the latest *departure time*?

This question arises when a user wants to depart as late as possible with a user-defined good-enough probability of making the deadline. For a path π with mean m_π and variance m_π the following relation holds.

$$v_\pi = \frac{1}{\varphi^2}\{(desired\ arrival\ time - departure\ time) - m_\pi\}^2, \tag{3.13}$$

where φ is the argument of the Gaussian CDF $\Phi(\cdot)$ that makes the $\Phi(\cdot)$ equal to the given probability of making deadline.

We are finding a path that maximizes the *departure time*, which is equivalent to finding a path that minimizes *desired arrival time - departure time.* We can rewrite the parabola equation (3.13) as

$$desired\ arrival\ time - departure\ time = m_\pi + \varphi\sqrt{v_\pi}$$

Thus, the optimization problem to solve is

$$\underset{\pi\in\Pi}{\operatorname{argmin}}\ m_\pi + \varphi\sqrt{v_\pi}.$$

The optimal path occurs among the extreme points of the convex hull composing every path point in mean-variance plane. This can be proven with similar arguments with those in Proof 3.3.2. Thus, we can apply the probe point method we developed in Section 3.3.2 to solve this new optimization problem. The difference is that the optimal parabola's curvature is fixed at $\frac{1}{\varphi^2}$and the apex is determined by the path point and the curvature. In the previous version of the problem the apex was fixed and the curvature of parabola was determined. The probe point is determined in the same way but the probe value is defined by $m_\pi + \varphi\sqrt{v_\pi}$. The probe point with a smaller probe value than that of the optimal parabola is

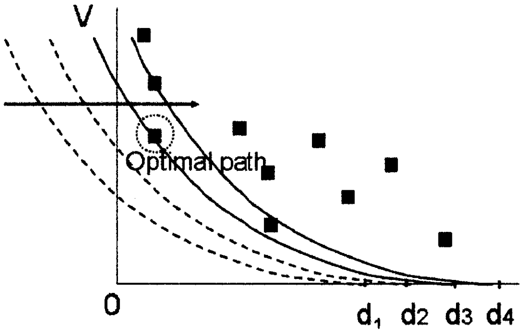


Figure 3-5: Illustration of the latest-departure path for a given probability guarantee. For a given probability of making deadline the curvature of the parabola is fixed. There is no path having the deadline as small as d_1 or d_2. The optimal path is indicated as green which has the smallest deadline among all the path, d_3. It has the latest departure time since *departure time = desired arrival time - deadline.*

kept. Intuitively, the optimization problem is to find the first path that intersects the parabola while we move the parabola from left to right (see Figure 3.3.2). The apex of the parabola stands for the deadline and the curvature stands for the probability of making the deadline. Thus, for the two optimization problems above, we optimize one when the other is fixed. We can also have numerous different optimality criteria that trades off between higher probability of making deadline and later departure time. Note that all the optimal paths for these different criteria occur among the extreme points, so that our methods can be used.

We can also use a similar technique as described in Section 3.3.2 to find a sub-optimal solution within a tolerable departure time range.

3.4 Synopsis

In this chapter, we presented an efficient stochastic shortest-path algorithm dealing with uncertainty in travel time. The algorithm uses a parametrical optimization method to find the exact optimal path. We showed the correctness of the algorithm and the running time of the algorithm.

Chapter 4

Stochastic Route Planning with Precomputation

This chapter describes a practical algorithm for stochastic motion planning and applications to route planning in the presence of traffic delays. This algorithm builds on the stochastic shortest-path algorithm explained in Chapter 3 and improves the query time by preprocessing. One main objective is to minimize the running time of the overall procedure at query time and hence the running times of these shortest-path subroutines are of high priority. We improve on the prior state of the art by designing, analyzing, implementing, and evaluating data structures that answer approximate stochastic shortest-path queries. For example, our data structure can be used to efficiently compute paths that maximize the probability of arriving at a destination before a given time deadline. The main theoretical result is an algorithm that, given a directed planar network with edge lengths characterized by expected travel time and variance, pre-computes a data structure in almost linear time such that stochastic approximate shortest-path queries can be answered in poly-logarithmic time (actual worst-case bounds depend on the specifics of the probabilistic model). The main experimental results are two-fold: *(i)* we provide methods to extract travel-time distributions from a large set of heterogenous GPS traces and we build a stochastic model of an entire city, and *(ii)* we adapt our algorithms to work for real-world road networks, we provide an efficient implementation, and we evaluate the performance of our method for the model of the aforementioned city, improving the query time by several orders of magnitude.

This chapter is organized as follows. We review prior work on stochastic motion planning in Section 4.1. We present an outline of our motion planning algorithm in Section 4.2. We describe practical considerations in Section 4.3 and experimental results in Section 4.4.

4.1 Background

4.1.1 Stochastic Shortest Paths

Consider a graph G with a given source node s and destination node t with stochastic edge delays w_e that come from independent distributions with means and variances (μ_e, τ_e) respectively. Denote the vector of edge delays by $\mathbf{w} = (w_1, ..., w_{|E|})$ and the corresponding vectors of edge means and variances by $\mu = (\mu_1, ..., \mu_{|E|})$ and $\tau = (\tau_1, ..., \tau_{|E|})$. Denote the incidence vector of a path by $\mathbf{x} \in \{0,1\}^{|E|}$ with ones corresponding to edges present in the path and zeros otherwise. Let the feasible set of s-to-t–paths be $\mathscr{F} \subset \{0,1\}^{|E|}$.

Given the uncertainty of edge delays, it is not immediate how to define a stochastic shortest path: is it the path that minimizes mean delay? Or should it minimize path variance or some other metric? Risk-averse users would naturally care about the variability in addition to mean delays. Two risk-averse models that have been previously considered are the mean-risk model and the probability-tail model, which we describe below.

The algorithms for both models make black-box calls to the underlying deterministic shortest-path problem:

$$\text{minimize } \mathbf{w}^T\mathbf{x} \quad \text{subject to } \mathbf{x} \in \mathscr{F}. \tag{4.1}$$

Probability-Tail Model In this model, we are given a deadline d and we seek the path maximizing the probability of arriving before the deadline: maximize $\Pr(\mathbf{w}^T\mathbf{x} \leq d)$ subject to $\mathbf{x} \in \mathscr{F}$. This model assumes normally-distributed edge delays in order to obtain a closed-form expression for the probability tail. In particular, when $\mathbf{w}$ is Gaussian, the problem is transformed into the following formulation [100]:

$$\text{maximize } \frac{d - \mu^T\mathbf{x}}{\sqrt{\tau^T\mathbf{x}}} \quad \text{subject to } \mathbf{x} \in \mathscr{F}. \tag{4.2}$$

Similarly to the mean-risk model, problem (4.2) can be solved exactly in time $n^{O(\log n)}$ [102] and it can be approximated as follows:

Theorem 5 ([100, Theorem 4.2]). *Suppose we have a δ–approximation algorithm for solving the deterministic shortest-path problem (4.1). For any $\varepsilon > 0$, the probability tail model (4.2) has a $\sqrt{1-\left[\frac{\delta-(1-\varepsilon^2/4)}{(2+\varepsilon)\varepsilon/4}\right]}$–approximation algorithm that calls the algorithm for the deterministic shortest-path problem polynomially many times in $1/\varepsilon$ and the input size, assuming the optimal solution to problem (4.2) satisfies $\mu^T x^* \leq (1-\varepsilon)d$ for a user-specified deadline d.*

A Mean-Risk Model In this model, we seek the path minimizing mean delay plus a risk-aversion coefficient $c \geq 0$ times the path standard deviation, more formally:

$$\text{minimize } \mu^T \mathbf{x} + c\sqrt{\tau^T \mathbf{x}} \quad \text{subject to } \mathbf{x} \in \mathscr{F}. \tag{4.3}$$

This is a non-convex combinatorial problem, which can be solved exactly by enumerating all paths that minimize some positive combination of mean and variance (the latter is a deterministic shortest-path problem with respect to edge lengths equal to the corresponding mean–variance linear combination) and selecting the one with minimal objective (4.3) (see [102, 100] for more details). The number of deterministic shortest-path iterations is at most $n^{O(\log n)}$ in the worst case [102]. Furthermore, there is a fully-polynomial-time approximation algorithm (FPTAS) for the problem, more precisely:

Theorem 6 ([100, Theorem 4.1]). *Suppose we have a δ–approximation algorithm for solving the deterministic shortest-path problem (4.1). For any $\varepsilon > 0$, the mean-risk model (4.3) can be approximated to a multiplicative factor of $\delta(1+\varepsilon)$ by calling the algorithm for the deterministic shortest-path problem polynomially many times in the input size and $1/\varepsilon$.*

In other words, one can use both exact ($\delta = 1$) and δ–approximate (for any $\delta > 1$) shortest-path subroutines and obtain a corresponding approximate risk-averse solution.

4.1.2 Approximate Distance Oracles

We point out one particular method, which is in some sense related to our construction. Geisberger, Kobitzsch, and Sanders [51] provide a heuristic shortest-path query method with edge lengths defined as a customizable linear combination of two edge length functions (parameter from a discrete interval of bounded size — which is why we cannot use their data structure).

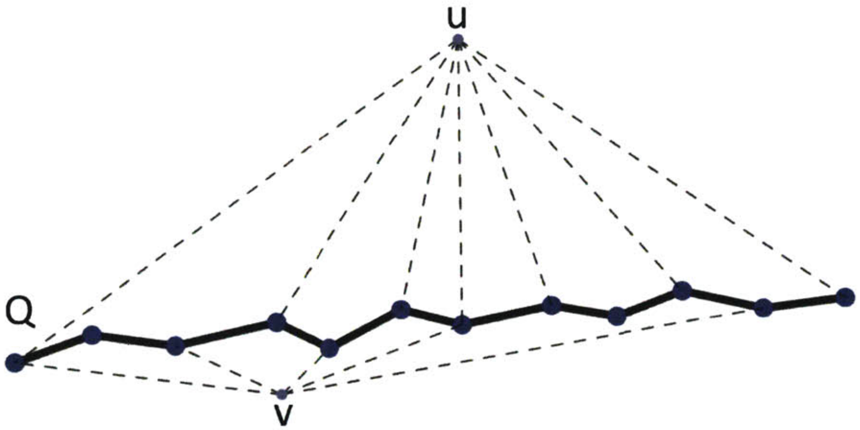


Figure 4-1: For each node v we have $O(1/\varepsilon)$ *portals* on each separator path Q such that the shortest path from v to each node $q \in Q$ is approximated within a $(1+\varepsilon)$–factor by going through one of the portals. Note that two nodes u and v may not necessarily use the same set of portals.

At a high level, Thorup's method works as follows. The preprocessing algorithm recursively *separates* the graph into subgraphs of roughly equal size. *Shortest paths* are used as separators. For planar graphs, it is possible to separate the graph into two roughly balanced pieces by using three shortest paths Q_1, Q_2, Q_3. Next, the preprocessing algorithm considers each node $v \in V$ and it computes $O(1/\varepsilon)$ *portals* for v on each separator path Q_i such that the shortest path from v to each node $q \in Q_i$ is approximated within a $(1+\varepsilon)$–factor by going through one of the portals. Since each node only needs to remember $O(1/\varepsilon)$ distances to portals per level, and since each node participates in $O(\log n)$ levels, the overall space complexity per node is bounded by $O(\varepsilon^{-1} \log n)$. For directed planar graphs, the construction is more complicated and the space and time complexities are slightly higher but the core idea is very similar.

4.2 Stochastic shortest-path queries using Distance Oracles

4.2.1 Overview

We provide a new stochastic motion planning algorithm and its implementation in a real traffic network. The main objective is to minimize the time for querying a motion planning path. Algorithms studied in [102, 83] find an exact solution by iteratively determining the parameter value k that governs the edge length $\ell(e) := k\mu(e) + \tau(e)$ in terms of mean μ and variance τ. Such parameter values depend on a specific problem, a deadline or a risk-aversion coefficient. We develop a method that finds a set of parameter values over the whole network (not depending on the specific problem) that provides guaranteed approximation accuracy for all choices of parameters *simultaneously*, while maintaining only a small set of parameters, which allows for small storage and efficient preprocessing. We preprocess the approximate optimal paths using the set of parameter values.

We implement the deterministic shortest-path algorithms required in [100] using a distance oracle, as described in the previous section. This allows for much faster query times. The preprocessing algorithm, while taking roughly linear time, is executed only once and the data structure can be computed on a fast machine (not necessarily the machine where the query algorithm is run on). One important technical difference compared to the method described in [83] is that we cannot use an adaptive algorithm at query time, since the preprocessing algorithm must prepare for each step that is executed at query time. In other words, the preprocessing algorithm must prepare for all possible query pairs $(s,t) \in V^2$, and, potentially, deadline parameter values $d \in \mathbb{R}$. While such preprocessing algorithms are standard for classical graph distances, non-trivial modifications are necessary for stochastic shortest paths.

4.2.2 Preprocessing Algorithm

We adapt the algorithms in Theorem 6 and Theorem 5 to our scenario, where we have access to an approximate oracle for the deterministic shortest-path problem with edge lengths $e \mapsto k \cdot \mu(e) + \tau(e)$ for some $k \in \mathcal{K}$. In previous work [100], the aim was a polynomial-time algorithm and shortest-path problems could be computed in roughly linear time for *any* value k. Since we aim at sublinear query time, the queries made to the oracle need to be non-adaptive, or at least restricted to a small set of possible queries $\mathcal{K}$. We can then precompute distance oracles for all these values $k \in \mathcal{K}$. On a high level, the preprocessing algorithm is rather simple.

```
Algorithm 6: PREPROCESS
Data: $G = (V,E)$ with lengths $\mu, \tau : E \to \mathbb{R}^+$
Result: distance oracles $orc_k$ for each $k \in \mathcal{K}$
compute the set $\mathcal{K}$;
for $k \in \mathcal{K}$ do
    $orc_k \leftarrow$ build (approximate) distance oracle for length function
    $e \mapsto k \cdot \mu(e) + \tau(e)$;
end
```

Note that this preprocessing algorithm can be parallelized in a straightforward way. Since there are no dependencies between the oracles constructed for different values of k, the preprocessing algorithm can also make use of a multi-core architecture.

As an immediate consequence, we obtain the following.

Claim 1. *The preprocessing time is bounded by $O(|\mathcal{K}| \cdot \mathcal{P})$, where $\mathcal{P}$ denotes the preprocessing time of the deterministic distance oracle.*

A main technical contribution of this work is showing how to compute (and analyze) this set $\mathcal{K}$. The size of $\mathcal{K}$ mostly depends on the user's choice of ε.

We run the deterministic preprocessing algorithm for the graph with edge lengths $\ell_k(e) = k \cdot \mu(e) + \tau(e)$ for $k \in \{L, (1+\xi)L, (1+\xi)^2 L, \dots U\}$ for ξ, L, U to be defined for each model. Also, the approximation parameter ε we use for the preprocessing algorithm in the distance oracle changes with the model.

A Mean-Risk Model Using a δ–approximate distance oracle, the end result is a $\delta(1+\varepsilon)$–approximation. Therefore, when asking for a $(1+\varepsilon)$–approximate answer, we internally set $\varepsilon_{int} := \varepsilon/3$, since, for $\varepsilon < 1/2$, we have that $(1+\varepsilon/3)^2 \leq 1+2\varepsilon/3+\varepsilon^2/9 \leq 1+\varepsilon$. In the following, we write ε instead of ε_{int}. Next, we set ξ, L, and U as follows.

$$\begin{aligned} \xi &= \sqrt{\frac{\varepsilon}{1+\varepsilon}} \\ L &= \min_{e} \tau(e) \\ U &= \sum_{e} \tau(e) \leq n \cdot \max_{e} \tau(e) \end{aligned}$$

Alternatively, U can be set to the largest possible variance of any path. Finally, we preprocess the distance oracle with $\ell_k(e)$ for all $k \in \mathscr{K} = \{L, (1+\xi)L, (1+\xi)^2 L, \ldots U\}$. We have that the total number of values k is at most

$$|\mathscr{K}| = \log_{1+\xi}(U/L) = O\left(\log\left(n\frac{\max \tau(e)}{\min \tau(e)}\right)/\sqrt{\varepsilon}\right).$$

As a consequence, we have the following lemma.

Lemma 3. *For any graph $G = (V,E)$ and for any two length functions $\mu : E \to \mathbb{R}^+$ and $\tau : E \to \mathbb{R}^+$, there exists a set $\mathscr{K}$ of size $|\mathscr{K}| = O\left(\varepsilon^{-1/2}\log\left(n\frac{\max \tau(e)}{\min \tau(e)}\right)\right)$ such that mean-risk shortest paths can be approximated by a shortest path in one of the graphs G^k, defined by G and the length function $\ell_k(e)$.*

Probability-Tail Model For the probability-tail model, we follow the same strategy as in the previous section, based on the proof of [100, Lemma 3.4], showing that, in the mean–variance plane, any fixed parabola (also termed *level set*) with apex $(d,0)$ can be ε–approximated by a piecewise linear curve (see [100, Figure 1(b)] for an illustration) using few line segments. We exploit this ε–approximation in our algorithm. Furthermore, we define level sets *irrespective* of the deadline d: we may actually define $\mathscr{K}$ for any value of d — for a given query deadline d', the actual parabolas are just shifted to the left or to the right. Each segment defines a slope value k and the union of all these slope values (their absolute value, actually) is our set of slopes.

We set ξ, L, and U as follows.

$$\begin{aligned}
\xi &= \varepsilon/2 \\
L &= \frac{2\min_e \tau(e)}{d_L} \\
U &= \frac{2\sum_e \tau(e)}{\varepsilon d_U} \leq \frac{2n\max_e \tau(e)}{\varepsilon d_U} \leq \frac{2n\max_e \tau(e)}{\varepsilon \min_e \mu(e)}
\end{aligned}$$

Although the user may choose to query the data structure for an arbitrarily large deadline d, we can give an upper bound on $d_L = \max d$ as follows. As soon as the deadline is so large that shortest paths do not change anymore, we do not have to consider larger values of d. In a straightforward way, d_L can be computed by computing all-pairs shortest paths for increasing values of d. Alternatively, a 2–approximation of the diameter can be found in linear time.

4.2.3 Query Algorithm

The simplest version just tries all values $k \in \mathscr{K}$.

Algorithm 7: QUERY

```
Data: origin $O$, destination $D$, distance oracles $orc_k$ for each $k \in \mathscr{K}$, and mean risk
      coefficient $c$ or deadline $d$ depending on the risk model $m$
Result: Approximate Optimal Path $p$
$p \leftarrow$ NIL;
for $k \in \mathscr{K}$ do
    $q \leftarrow find\_path(O,D,orc_k)$ ;
    if $(m = meanRisk\ and\ q.\mu + c\sqrt{q.\tau} < p.\mu + c\sqrt{p.\tau})\ or\ (m = probTail\ and$
    $\frac{d-q.\mu}{q.\tau} > \frac{d-p.\mu}{p.\tau})$ then
        $p \leftarrow q$;
    end
end
return $p$;
```

Claim 2. *The query time is bounded by $O(|\mathscr{K}| \cdot \mathscr{Q})$, where $\mathscr{Q}$ denotes the query time of the deterministic distance oracle.*

Combined with Claim 1, we obtain our main theorem.

Theorem 7. *There is a data structure that can be computed in time $O(|\mathscr{K}| \cdot \mathscr{P})$, occupying space $O(|\mathscr{K}| \cdot \mathscr{S})$ answering approximate stochastic shortest-path queries in time $O(|\mathscr{K}| \cdot \mathscr{Q})$, where $\mathscr{P}, \mathscr{S}, \mathscr{Q}$ denote the preprocessing, space, and query complexities of the deterministic distance oracle, respectively.*

In the practical part, we also have a heuristic that binary searches for the optimal value in $\mathscr{K}$, resulting in faster query times.

Plugging in the distance oracle of Thorup [123], we obtain our main result. We state it in the mean-risk model. An analogous bound holds for the probability-tail model.

To apply [123, Theorem 3.16], we set the largest integer length $N^{(k)} := \max_{e \in E} k \cdot \mu(e) + \tau(e)$ for a preprocessing step $k \in \mathscr{K}$. Also, let $\bar{\tau} = \frac{\max \tau(e)}{\min \tau(e)}$.

Corollary 1. *For any directed planar graph, there exists a data structure that can be computed in time $O(n(\log(n\bar{\tau}))(\lg(nN))(\lg n)^3 \varepsilon^{-5/2})$, occupying space $O\left(n \cdot \varepsilon^{-3/2}(\lg n)(\lg(nN))(\log(n\bar{\tau}))\right)$ such that approximate stochastic shortest-path queries can be answered in time $O(\varepsilon^{-1/2} \log(n\bar{\tau}) \lg\lg(nN) + \varepsilon^{-3/2} \log(n\bar{\tau}))$.*

The space can be reduced at the cost of increasing the query time by using the construction in [70]. In particular, for scenarios where memory is scarce (such as a mobile device), one can obtain a *linear-space* data structure. Note that the preprocessing algorithm can be executed on a fast computer and not necessarily on the mobile device. The query time remains poly-logarithmic.

4.3 Practical Considerations

We adapt our method for planar networks so that it can efficiently handle the actual road networks we use in our experiments.

4.3.1 Non-planarities

Our network is not exactly planar and many real-world road networks may be non-planar [43]. The internal algorithms of the distance oracle use separator paths, which are sensitive to non-planarities. We propose to address most of these non-planarities as follows (inspired by similar constructions for minor-free graphs [11]). On the highest level of the recursion, instead of using two shortest paths to separate the graph, *(i)* we interpret the set of express ways (cf. highways) as a set of separator paths, *(ii)* we compute portals for each separator path Q and each node v, (distances to portals are computed with respect to the entire graph) and *(iii)* we remove the set of express ways from the graph. The highest level of the recursion handles all paths that use any part of an express way. The remaining graph has only few further crossings.

Turn restrictions For some intersections, particularly in urban settings, it may not be possible to make a left turn due to a *turn restriction*, or it may just be more time-consuming to make a left turn, which can be modeled by a *turn cost*. Turn costs can be taken into account by modeling each road segment by a node of the graph, and then connecting only those nodes whose corresponding road segments can be traversed in sequence [24, 132]. Such a transformation increases the number of nodes [53] and, more importantly for our oracles, it also violates planarity. Planarity is not just violated in a moderate way but the violation is actually such that it cannot possibly be fixed. If arbitrary turn restrictions were allowed, we could just *(i)* draw any graph in the plane (for example the large-girth graphs in [124] or the expander graphs in [120]), *(ii)* add a node for each intersection, *(iii)* apply left-turn and right-turn restrictions (i.e. apply the supposedly-existing transformation without violating planarity), *(iv)* compute a planar distance oracle [123], and *(v)* store only the distance labels for the actual nodes of the graph. If the transformation in step *(iii)* could be done, the resulting oracle would contradict known lower bounds. In our implementation, we connect each node to additional portals as follows: on any separator path, we deem each turn-restricted intersection as a portal. The upper bound of $O(1/\varepsilon)$ portals per separator path cannot be guaranteed anymore. The bound in the implementation is $O(1/\varepsilon+R)$, where R is the number of intersections with turn restrictions on a given separator path.

4.3.2 Directions

Realistic modeling of traffic requires us to consider directed graphs.[1] The existing implementation [94] is for undirected graphs only and the directed oracle is significantly more complicated [123]. Existing methods for directed road networks [52, 115, 17, 18] usually do not have theoretical worst-case bounds close to [123] (some theoretical results are known [10, 9]). For our scenario, we integrate the undirected implementation into our code. While our theoretical results hold for directed networks, our experimental evaluation is for undirected networks.

4.3.3 Reduced Set of Portals on Separator Paths

On the highest level, we cannot use planar-graph algorithms to efficiently compute portals and distances to portals for each node. Instead, we propose to relax the approximation guarantee in a controlled way. We heuristically treat each separator path (or express way) as in Algorithm 8.

4.3.4 Deadline Values

In the probability-tail model, the user may specify an arbitrary deadline d. As a consequence, the ratio of the smallest and the largest possible d can be arbitrary large. As argued in the theoretical section, this can be controlled for effectively. In practical implementations, one could also limit the range of user input values by restricting the deadline values available.

4.4 Experiments

We evaluate an implementation of our algorithm on a real-world network dataset. We show that the query time is several orders of magnitude faster than the previous algorithms, and

[1] The shortest-path problem for directed graphs is strictly more general. In a straightforward way, any undirected graph can be represented by a directed graph with bidirectional edges. For arbitrary directed graphs, even if they are guaranteed to be sparse, not much is known with respect to shortest-path queries, while the undirected variant is quite well understood.

```
Algorithm 8: PORTALS$(H,Q,k)$
Data: a path $Q$ and a parameter $k$
Result: a set of portals for each $v \in V(H)$
SINGLESOURCESHORTESTPATHS$(Q)$ ;
connect dummy source to each $q \in Q$;
computes first portal $q_0(v)$ and distance $d(v,q_0(v))$ for each $v \in V(H)$;
for $q_i \in Q$ do
    for $j \in \{0,1,2,\ldots k-1\}$ do
        $Q^{(j)} \leftarrow$ subset of nodes $q_i$ where $i = j \mod k$ ;
        SINGLESOURCESHORTESTPATHS$(Q^{(j)})$ ;
        dummy source connected to all $q \in Q^{(j)}$;
        computes another portal $q_j(v)$ and distance $d(v,q_j(v)$ for each $v \in V(H)$;
        if $(1+\varepsilon)d(v,q_j(v)) > d(v,q_i(v))+d(q_i(v),q_j(v))$ for all portals $q_i(v)$ added
        previously then
            add portal;
        end
    end
end
```

we also show that the approximate path is close to the optimal path.

4.4.1 Setup

The tests were run on a machine with 2 GB of main memory and an Intel Xeon CPU 1.86GHz processor.

Our graph represents a city-scale network with 40,000+ nodes and 70,000+ edges. As described in the previous section, we adapted the worst-case-proven algorithm and tailored it to fit our real-world scenario. We implemented the distance oracle for undirected graphs based on the implementation of [94].

Since our network is mostly bidirectional (meaning that there are only few one-way streets), our graph could potentially be considered undirected. However, in time-dependent scenarios, the traffic may still flow in one direction, while a road may potentially be jammed in the other direction. Using the following heuristic, we can use the undirected variant of the distance oracle without significant impact on the path quality. To compile the different travel time distribution of bidirectional road segments, we built distance oracles for two different edge length functions: one with *into-the-city-center* driving time statistics and one with *out-of-the-city-center* driving time statistics. The orientation of an edge with respect to the city center was used as the reference to indicate which of the two travel time statistics should be used when constructing the distance oracle. At query time, for example, if the route query overall corresponds to a *into-the-city-center* movement, the distance oracles built based on the *into-the-city-center* statistics are used.

We examined the preprocessing time, query time, and approximation ratio for different ε values for the two different models: Specifically, we used $\varepsilon \in \{0.1, 0.2, 0.3, 0.4, 0.5, 1\}$ for the Mean-Risk model, and $\varepsilon \in \{0.5, 1\}$ for the Probability-Tail model.

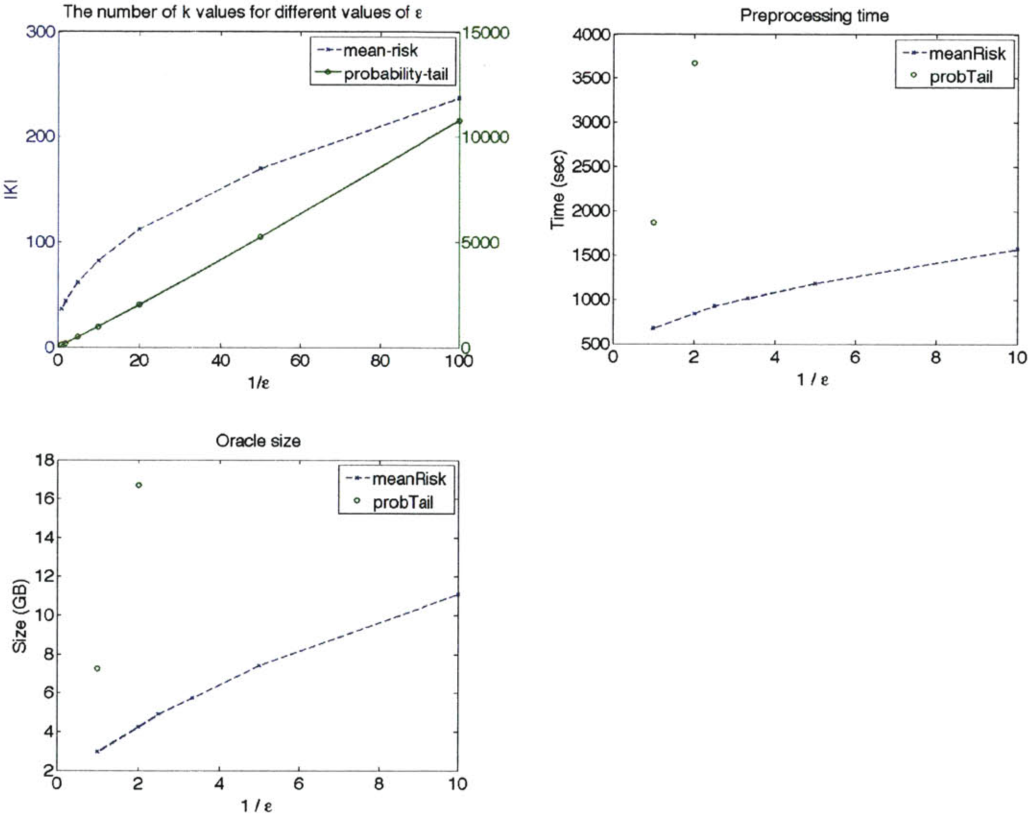


Figure 4-2: The number of k values needed for the road network. this figure illustrates the growth of $|\mathscr{K}|$ for the mean-risk and the probability tail model with respect to the user-specified level of accuracy ε (Left), Preprocessing time for each ε (Middle), For each ε value ($1/\varepsilon$ on the x–axis), we report the total storage requirement for all the distance oracles (y–axis) (Right).

4.4.2 Preprocessing

Depending on the user's desired accuracy level ε, we find the values ξ, L, and U (as outlined in the theoretical part). The sizes of the set $\mathscr{K}$ for different values of ε are shown in Figure 4-2 (Left). The preprocessing time depends mostly on $|\mathscr{K}|$ (and therefore on ε). For each value of ε, we used $c \in \{0, 0.5, 1, \cdots, 6\}$ for the Mean-Risk model, and $d \in \{1.1 \cdot m^*, 1.2 \cdot m^*, \cdots, 2 \cdot m^*\}$ for the Probability-Tail model, where m^* is the minimum expected travel time for a given origin-destination pair.

Computation time The preprocessing time for different ε values is shown in Figure 4-2 (Middle). For example, for the Mean-Risk model, if we set $\varepsilon = 0.1$, 82 different k values are used. The overall preprocessing time is 5,159 seconds. The average preprocessing time for one distance oracle is roughly 63 seconds.

Note that the number of distance oracles to pre-process only depends on the desired accuracy of the solution paths defined by ε. It is independent of the users' other query parameters, such as origin, destination and the risk-aversion coefficient c in the Mean-Risk model or the deadline d in the Probability-Tail model.

Data space Depending on ε, the overall space consumption varies greatly (see Figure 4-2 (Right)), mostly due to two reasons. First, the number of k values increases as the accuracy is increased (meaning that ε is decreased), see Figure 4-2 (Left), and second, the number of connections in the distance oracle grows linearly with $1/\varepsilon$.

4.4.3 Path Queries

To measure the query time, we compute stochastic shortest paths between 1,000 random origin-destination (OD) pairs. To evaluate the quality and accuracy of the reported paths, we query 1,000 random OD pairs.

Query time Figure 4-3 shows the time required to answer a route query for different values of ε. The query time decreases as ε increases. We implemented [83, Algorithm 1] and plotted the query time result in Figure 4-3 as a comparison to our method in this chapter.

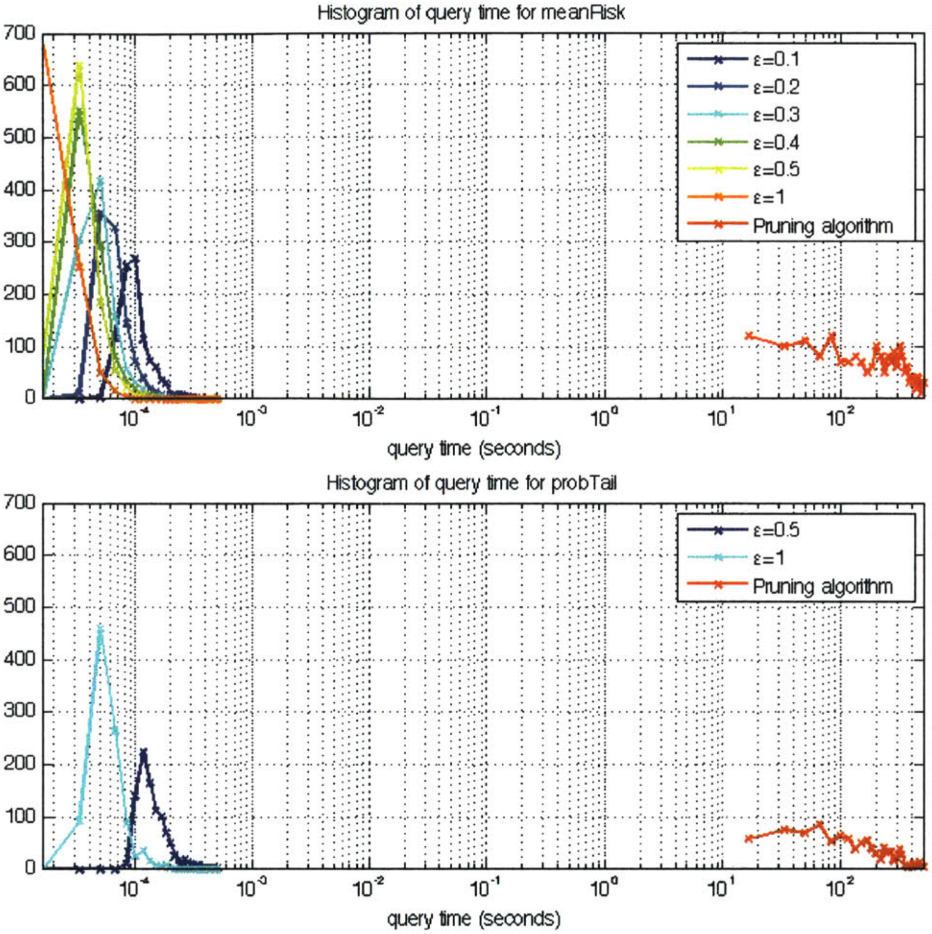


Figure 4-3: The histogram of the query time for 1,000 random OD-paris. 'Pruning algorithm' refers to Algorithm 1 in [83]. Our new method is faster by several orders of magnitude. The speed-up is roughly from tens of seconds or several minutes to tens of or hundreds of microseconds.

Approximation quality We compare the quality of the path output by our algorithm to the optimal path length, computed by the algorithm described in [83]. As a quality measure, we list the approximation ratio, defined as the difference between the optimal solution and our solution, divided by the optimal solution (see Figure 4-4). Our worst-case guarantee says that this ratio can never be larger than the user-specified level of accuracy ε. Our experimental results show that the ratio is smaller than ε, and, moreover, the ratio is substantially smaller than ε. Even for relatively large values such as $\varepsilon = 1$, we still obtain an approximation ratio less than 0.02. Based on this experimental result, we suggest using

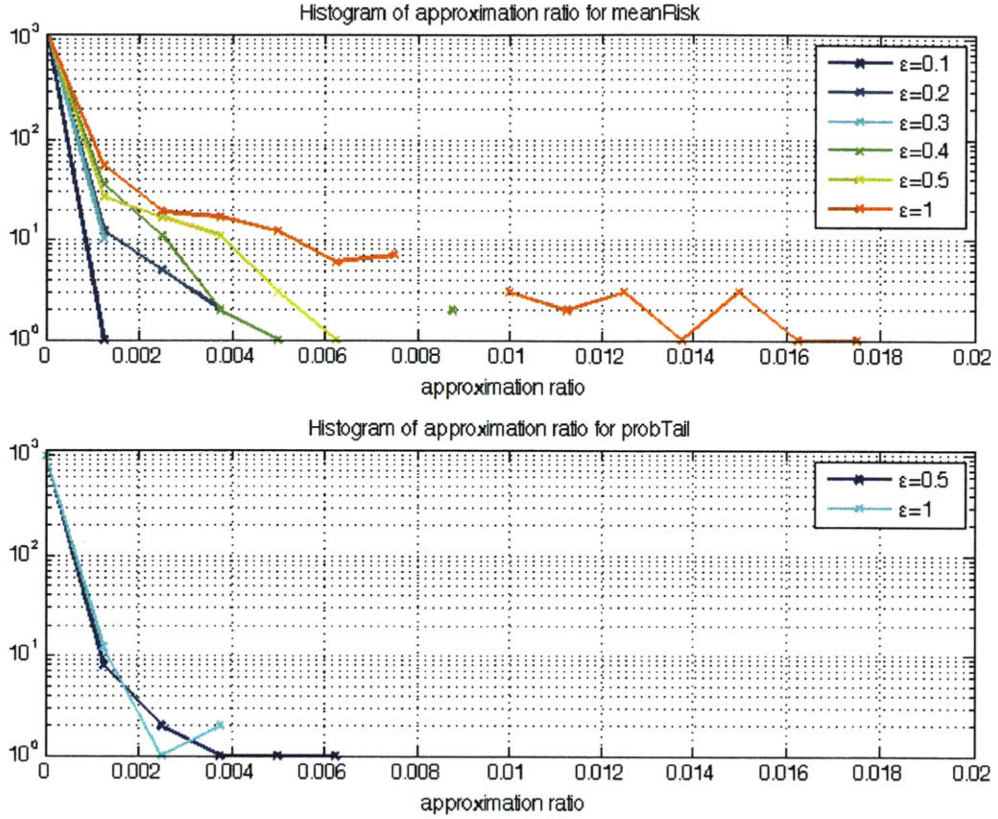


Figure 4-4: The histogram of the approximation ratio for 100 random OD-paris. We used $c \in \{0, 0.5, \cdots, 6\}$ for the Mean-Risk model and $d \in \{1.1 \cdot m^*, 1.2 \cdot m^*, \cdots, 2 \cdot m^*\}$ for the Probability-Tail model, where m^* is the minimum expected travel time for a given origin-destination pair.

larger values of ε, saving both computation time (both preprocessing and query time) and storage (as outlined in the previous section).

4.4.4 Path Examples

We developed a web interface that responds to user inputs. Users can select the origin and destination, as well as the travel time of the day and the day of the week. The system computes the stochastic shortest path and displays it on the map. Figure 4-5 shows our different solution paths for different deadlines for the Probability-Tail model with $\varepsilon = 0.1$.

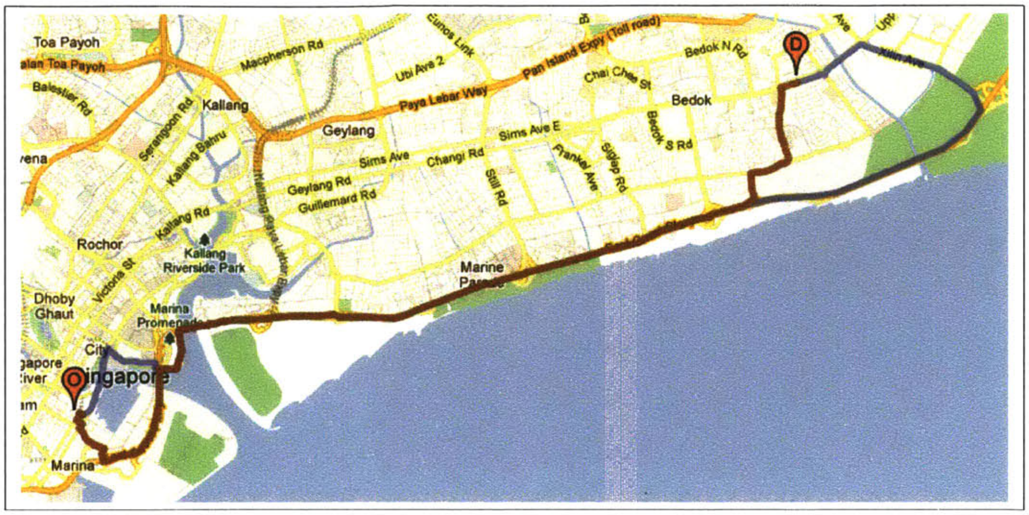


Figure 4-5: Path query example for maximizing the probability of reaching the destination within a user-specified deadline. Our system finds such a path within milliseconds for a city-sized network. When a user at origin 'O' wants to get to destination 'D' within 30 minutes between 5pm and 6pm on a weekday, the blue path offers the best chance of reaching the destination within the deadline. However, when the deadline changes to 20 minutes, the blue path has less than a 50% chance for making the deadline. In this case, the red path is the best route, offering a 88.5% chance for on-time arrival.

4.5 Synopsis

In this chapter, we showed an improved algorithm upon the state of the art in stochastic motion planning, providing a method that can answer stochastic shortest-path queries in almost logarithmic time using a data structure that occupies space roughly proportional to the size of the network. Our method has proven worst-case guarantees and it is applicable to real-world scenarios. Applied to traffic navigation, we obtain a method that can efficiently find a path with almost maximum arrival probability given a user-specified deadline.

Chapter 5

Stochastic Multi-Agent Route Planning

In this chapter, we present route planning algorithms for multiple agents in a graph with stochastic travel times. The goal is to provide a route plan for a group of agents to visit multiple locations such that the route is optimally selected to maximize the probability of completing the path within a given deadline. The algorithm aims to find the best agent, the best resources and the best path through all the visit locations guaranteeing the highest probability for achieving the goal.

Though much research has been done in stochastic shortest-path problems, the existing algorithms focus on the single-origin single-destination problem for one agent. This chapter formulates a general framework for the stochastic shortest-path problem with visit node constraints designed to achieve a specific goal with multiple agents, multiple resources, and multiple destinations. The constraints are defined by a set of sequences of nodes to be visited. Given predetermined constraints, our motion planning problem consists of finding the best agents, resources, and destinations, and the path through a sequence of nodes representing them.

The single-source single-destination stochastic shortest-path algorithms cannot be directly applied to the problems with multiple agents and multiple visit locations since the optimality sub-structure does not hold. In this chapter, we provides an efficient algorithm for such complex problems using the building blocks of a stochastic shortest-path algorithm presented in [83]. The technique in this chapter solves the problem at the same level of complexity as solving the single-origin single-destination problem by parallelization.

We demonstrate the algorithm by a Web-based traffic navigation guide system and evaluate the algorithm's performance.

The main contributions of this chapter are as follows:

1. Formulating the multi-agent, multi-resource, multi-hop navigation problem as a graph search problem with visit node constraints.

2. Developing a solution to the stochastic shortest-path problem with visit node constraints.

3. Introducing a decentralized algorithm for multi-user congestion-aware routing that is guaranteed to achieve the social optimum–where the goal is to minimize the overall travel time in the system.

4. Implementing and evaluating the algorithm in a real road network with real historical travel time data.

In Section 5.1, we offer a solution to the Stochastic Shortest Path with Fixed Node Sequence (SSPFNS) problem, where an ordered list of nodes to be visited is specified. In Section 5.2, we solve the Stochastic Shortest Path with General Constraints (SSPGC) problem, where constraints describe complicated combinations of node visit sequences. The SSPGC problem covers more general problems, including multiple-agent, multiple-resource, and multi-hop routing applications. In Section 5.3, we discuss a couple of extensions to the problems: First, we give a method of incorporating the visit cost into the algorithms. Second, we provide a method for considering statistical correlations between edges. Section 5.4 shows experimental results.

5.1 Multi-Hop: Stochastic Shortest Path Through a Fixed Node Sequence (SSPFNS)

5.1.1 Problem Formulation

The SSPFNS problem is defined as finding the optimal path (such as the path that maximizes the probability of arrival at the destination within a given deadline) for visiting nodes in a predefined sequence $m_1, m_2, \cdots, m_M$, where m_1 models the original location of an agent and m_M is the destination, with $m_2, \cdots, m_{M-1}$ modeling the locations of required intermediate visits. We refer to these nodes as *visit nodes*. The travel cost for edges is modeled as a normal distribution as in Section 3.1. The cost for visiting a node is assumed to be 0, but we will discuss how to consider the visit time in Section 5.3.1. This problem formulation captures many crucial applications such as planning a delivery truck's route to maximize the probability of delivering all the delivery goods within the deadline.

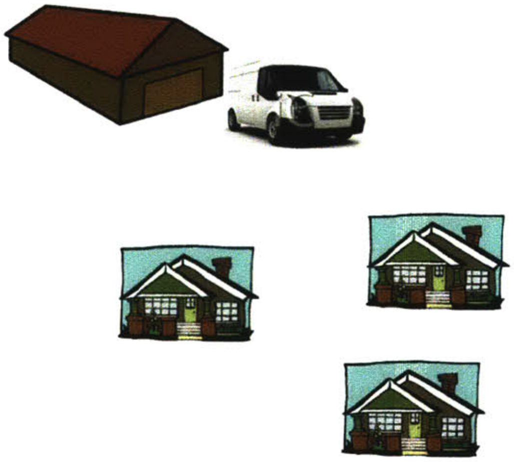


Figure 5-1: The delivery truck wants to find the path to maximize the probability of finishing the work by 5pm.

Let the feasible path set P_f be the set of all the paths that satisfy the fixed node sequence. Each path of road network G is shown as a point and each path in the feasible path set is shown in red in Figure 5-2. The problem is finding the best path in the feasible path set.

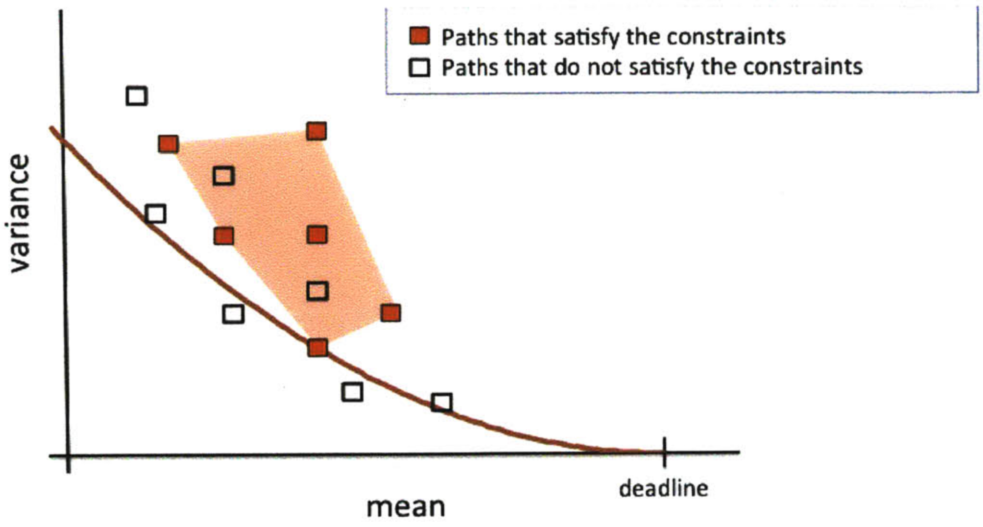


Figure 5-2: Red points represent the paths satisfying the constraints. The important observation here is that the optimal path with constraints occurs among the extreme points of red squares.

5.1.2 Solution Method

The first important property for an efficient algorithm is that the optimal path for the SSPFNS problem is one of the extreme points of the convex hull created by the feasible path set in the mean-variance plane. Moreover, the extreme points are represented by a λ-optimal solution satisfying the fixed node sequence for a certain λ value.

Finding the λ-optimal Path Satisfying the Fixed Node Sequence

The second property for the solution method is that whereas the optimal path is not a concatenation of the optimal paths for each hop of the fixed visit sequence, the λ-optimal solution *is* a concatenation of the λ-optimal paths for each hop of that sequence.Thus, the same parametric optimization and pruning methods developed in [83] can be used to solve the SSPFNS problem, except that now we focus only on the paths that satisfy the constraints.

These two properties are proven in Section 5.1.3. Using these properties we develop the SSPFNS algorithm as shown in Algorithm 9,

Algorithm 9: Find-Optimal-Path-FNS

```
Data: Graph and fixed node sequence
Result: optimal path from the origin to destination through the fixed node sequence
optimalPath := NULL;
while ($\lambda$:=Find-Next-$\lambda$()$\neq$NULL do
    $\pi$ := Find-$\lambda$-Optimal-Path-FNS($\lambda$, fixed node sequence);
    optimalPath := Update-Regions-and-Optimal-Path($\pi$);
end
return optimalPath
```

Algorithm 10: Find-λ-Optimal-Path-FNS

```
Data: $\lambda$, fixed node sequence
Result: $\lambda$-optimal path through visit nodes
for each pair of subsequent visit nodes $n_i$ and $n_{i+1}$ do
    $\pi_{n_i \to n_{i+1}}$ := Find-$\lambda$-optimal path($\lambda$, $n_i$, $n_{i+1}$);
end
Find the concatenation of the found paths: $\pi_{n_i \to n_{i+1}}, \forall i$;
```

5.1.3 Correctness

First, we prove that the optimal path occurs among λ-optimal paths from the feasible set. Second, we show that the concatenation of each pair of subsequent nodes' λ-optimal paths is the λ-optimal path from the feasible set.

Theorem 8. *The optimal path occurs among the extreme points of the convex hull including every path point satisfying the visit node sequence in the mean-variance plane.*

Proof. Let point P on the mean-variance plane represent the optimal path satisfying the visit node sequence. Then, every other path point satisfying the constraints must be inside the parabola that passes through P having its apex at $(deadline, 0)$. Since the parabola is convex, P must be an extreme point for the convex hull including all the path points satisfying the visit node sequence. □

As shown in [83], an extreme point of a set of path points corresponds to the λ-optimal path for the set of paths for a certain λ. The method of finding the λ-optimal path is described in Algorithm 10. Theorem 9 proves that this method is correct.

Theorem 9. *Let the edge weight be given by mean $+ \lambda \cdot$ variance. The minimum-cost path that visits all the visit nodes is the concatenation of all the minimum-cost paths between each pair of subsequent visit nodes.*

Proof. Let n_1, n_2 ,$\cdots$,n_m be the nodes to be visited. Let path $\pi^*_{n_i \to n_{(i+1)}}$ be the minimum-cost path from n_i to $n_{(i+1)}$. Let a, b, c be the three subsequent visit nodes. The minimum-cost path from a to c via b is the concatenation of the minimum-cost paths from a to b and b to c. Thus, the minimum-cost path that visits all the intermediate nodes is the concatenation of $\pi^*_{n_1 \to n_2}$, $\pi^*_{n_2 \to n_3}, \cdots , \pi^*_{n_{m-1} \to n_m}$. □

From Theorems 8 and 9 we know that there exists a λ such that the concatenation of λ-optimal paths between the subsequent visit nodes in the sequence is the optimal path for that sequence.

5.1.4 Complexity Analysis

The difference between Algorithm 1 and Algorithm 9 is that the building block Algorithm 2 is replaced with Algorithm 10 that includes $M-1$ iterations of Algorithm 2, where M is the number of visit nodes. Thus, the complexity of the SSPFNS algorithm is at most $M-1$ times greater than that of the SSP algorithm in [83]. The algorithm can run in parallel since there is no computational dependency between finding λ-optimal paths for each pair of subsequent visit nodes. Thus, the running time is the same as the SSP algorithm when we utilize $M-1$ computation nodes.

5.2 Multi-Agent and Multi-Resource: Stochastic Shortest Path with Generalized Constraints (SSPGC)

5.2.1 Problem Formulation

In this section, we define the Stochastic shortest-path problem with Generalized Constraints. Whereas the SSPFNS problem considers only a fixed node sequence, the SSPGC problem considers a more general class of visit node constraints.

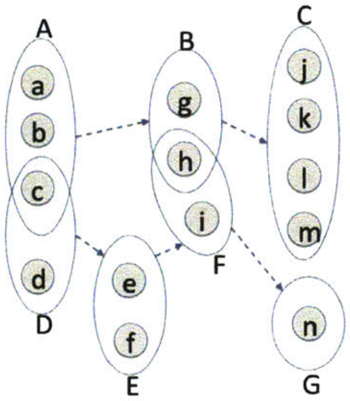


Figure 5-3: The graph shows general constraints.

Figure 5-3 illustrates an example of visit node constraints, where one node in a group (indicated in oval) should be visited in the order described by arrows. For example, a visit sequence $a \rightarrow g \rightarrow k$ is a valid sequence satisfying the constraints. Let $\wedge$ be the *ordered and* operator and $\vee$ be the *or* operator. Distributive and associative properties hold for both $\vee$ and $\wedge$, but the commutative property holds only for $\vee$ since the order of the two items for a $\wedge$ matters. Using these operators, we can express the constraints as follows: $((a \vee b \vee c) \wedge (g \vee h) \wedge (j \vee k \vee l \vee m)) \vee ((c \vee d) \wedge (e \vee f) \wedge (h \vee i) \wedge (n))$. If we apply the distributive law for all the terms and transform the constraints into $\vee$'s of sequences with only $\wedge$'s, each sequence with only $\wedge$'s represents a valid visit sequence. We refer to the visit sequence with only $\wedge$'s as an *elementary sequence*.

Remark 2. *$\wedge$ and $\vee$ notations can describe other constraints too. For example, constraints of visiting a set of nodes in any order can be described by enumerating possible permutations of nodes with $\wedge$ and $\vee$. If all the nodes in Group E in Figure 5-3 should be visited, in any order, we can express it as $(e \wedge f) \vee (f \wedge e)$ as drawn in Figure 5-4.*

Whereas the SSPFNS problem's fixed node sequence can represent only one *elementary sequence*, the constraints in SSPGC represent general constraints that can model many applications with multiple agents, multiple resources, and multiple destinations. Functionally equivalent agents' locations can be modeled as nodes in the same group. Similarly, functionally equivalent resources' locations can be modeled as nodes in another group. The constraints can be modeled as a sequence of these groups, meaning that any one from

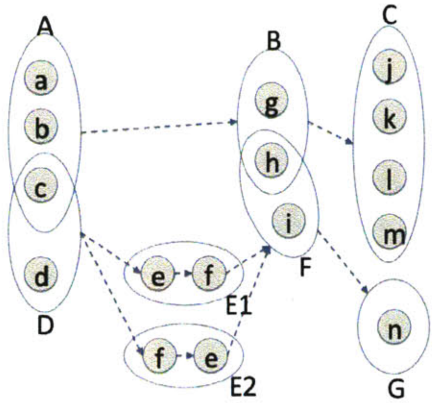


Figure 5-4: This figure illustrates how we can decompose the constraints of a group of nodes being visited in any order into $\wedge$ and $\vee$.

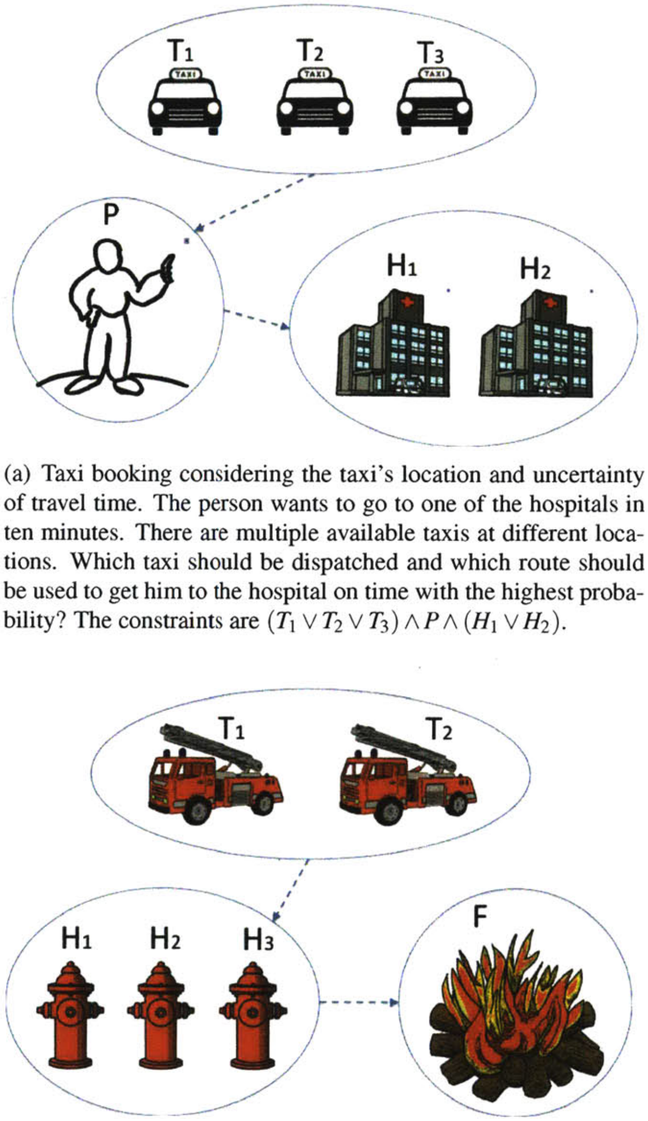
the same group can be chosen. The goal is to find the best choice of the agents, resources, destinations, and routes for the agents. Figures 5-5(a) and 5-5(b) illustrate examples.

5.2.2 Solution Method

We can think of a naive solution that finds the optimal path for each possible elementary sequence using the SSPFNS algorithm developed in Section 5.1 and compares them to find the best path among them. However, this is computationally expensive since the number of such sequences grows exponentially with the number of total nodes of constraints, M. We develop an efficient algorithm in this section.

Union of Extreme Points

Paths satisfying different elementary sequences are shown in different colors (red and green) in Figure 5-6(a). Each parabola represents the level set of the optimal path for each elementary sequence. The best path among the optimal solutions for each elementary sequence is the optimal solution for the general constraints. As we pointed out, a naive solution treating each elementary sequence differently is not desirable. The intuition behind the efficient solution is that we see the optimal solution for general constraints as the best solution among the path points that satisfy either of the elementary sequences. It is equivalent to finding the optimal solution from the convex hull including the union of path

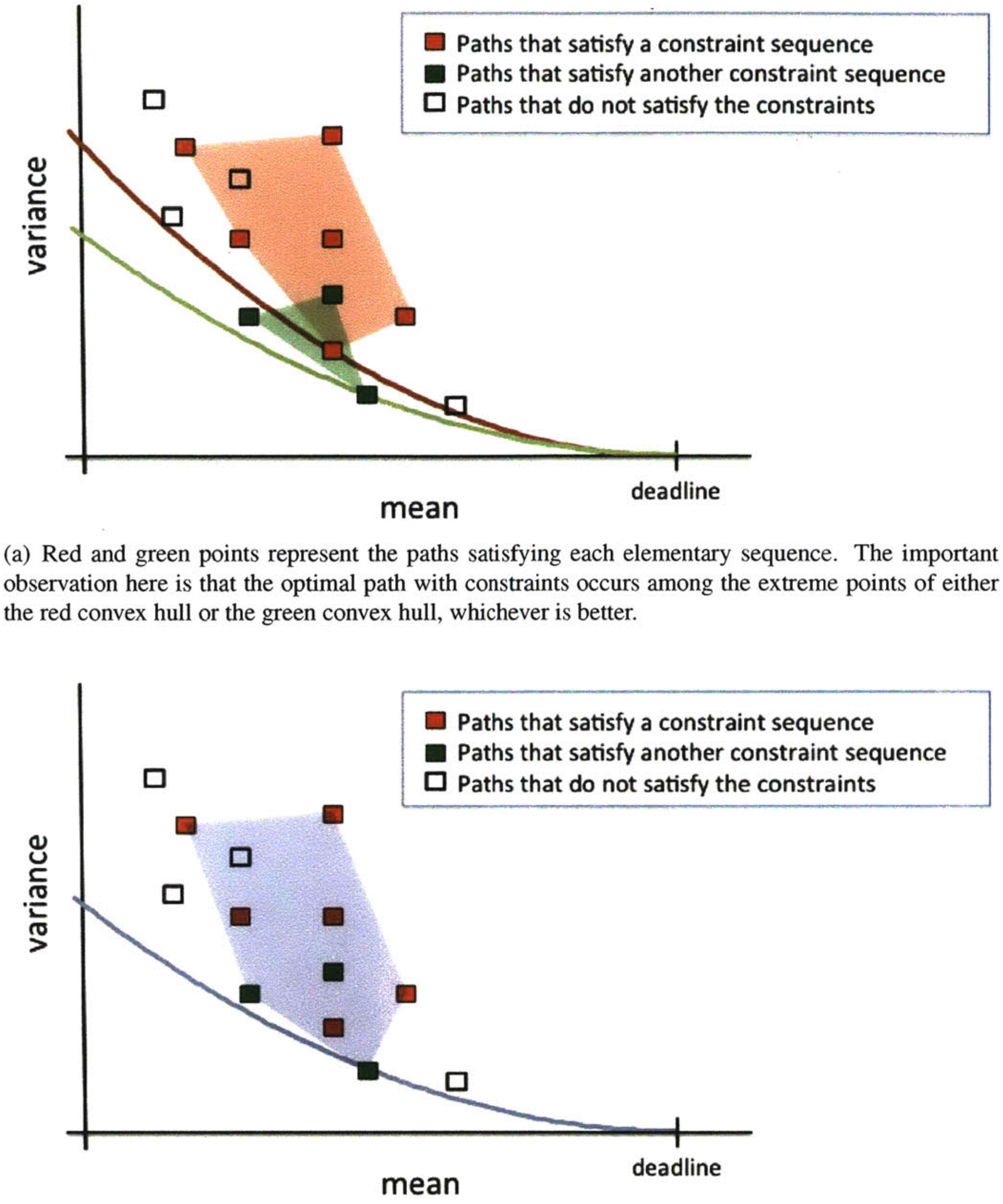


(a) Taxi booking considering the taxi's location and uncertainty of travel time. The person wants to go to one of the hospitals in ten minutes. There are multiple available taxis at different locations. Which taxi should be dispatched and which route should be used to get him to the hospital on time with the highest probability? The constraints are $(T_1 \vee T_2 \vee T_3) \wedge P \wedge (H_1 \vee H_2)$.

(b) A fire has occurred and it should be extinguished within 5 minutes. There are a few fire hydrants. A fire truck should visit one of the fire hydrants on the way to the fire. Which fire truck and which hydrant should be chosen and which route should be taken to maximize the probability of getting to the fire in time? The constraints are $(T_1 \vee T_2) \wedge (H_1 \vee H_2 \vee H_3) \wedge F$.

Figure 5-5: Example applications

points from each elementary sequence as visualized in Figure 5-6(b). The important fact is that we do not need to find the optimal solution for the elementary sequence separately. We can find the optimal solution for the general constraint at once, if we can find the λ-


🟥 Paths that satisfy a constraint sequence
■ Paths that satisfy another constraint sequence
□ Paths that do not satisfy the constraints
variance
mean
deadline
🟥 Paths that satisfy a constraint sequence
■ Paths that satisfy another constraint sequence
□ Paths that do not satisfy the constraints
variance
mean
deadline

(a) Red and green points represent the paths satisfying each elementary sequence. The important observation here is that the optimal path with constraints occurs among the extreme points of either the red convex hull or the green convex hull, whichever is better.

(b) The blue convex hull includes every path point satisfying the constraints. The key for the efficient algorithm is that we can do the λ-optimal search for the union of red and green points rather than doing the search for each color separately.

Figure 5-6: Illustration of optimal path in mean-variance plane

optimal path for the constrained network. We develop a method to find the λ-optimal path satisfying the general constraints in the following section.

Finding the λ-optimal Path for General Constraints

We introduce a hierarchical approach to find the λ-optimal path for general constraints. At the first level, we find the λ-optimal path for all the $\wedge$ constraints between every pair of subsequent visit nodes, called *atomic constraints*. Second, we create a new graph, called a *constraint graph*, whose nodes include only visit nodes and whose edges are associated with the λ-optimal path cost found at the first level. Then, we find the minimum-cost path on the constraint graph.

Definition 2. *An* atomic constraint *is the visit order requirement between two visit nodes. If we have a $\wedge$ constraint between two groups of nodes each having m and n nodes, respectively, we have $m \cdot n$ atomic constraints.*

Definition 3. *A* constraint graph *is a graph with each edge representing an atomic constraint.*

Complicated $\wedge$ and $\vee$ constraints can be transformed into a constraint graph with visit nodes and atomic constraints as edges.[1] We add virtual origin and destination nodes to the constraint graph with edges connecting the virtual origin to the first group of nodes and connecting the last group of nodes to the virtual destinations with associated weight 0. The problem is now to find the SSP with the intermediate node constraints.

We transform the constraints described with $\wedge$ and $\vee$ constraints in Figure 5-3 to a constraint graph shown in Figure 5-7, where edges represent the atomic constraints. The red nodes are the virtual origin and destination that are introduced with the red edges having 0 weights. The blue line is the minimum-cost path between the two nodes where the edge weight is $mean + \lambda \cdot variance$. The finding of each pairwise minimum-cost path can be computed in parallel. Once the pairwise minimum-cost paths are computed, the λ-optimal path from the virtual origin to the virtual destination can be found easily using Dijkstra's algorithm. Finding the minimum-cost path for all the atomic constraints requires at most $O(M^2)$ Dijkstra's algorithm executions. We can run these executions in parallel.

[1] If the constraint graph has cycles, then we need to first transform the graph into a graph that does not have any cycles.

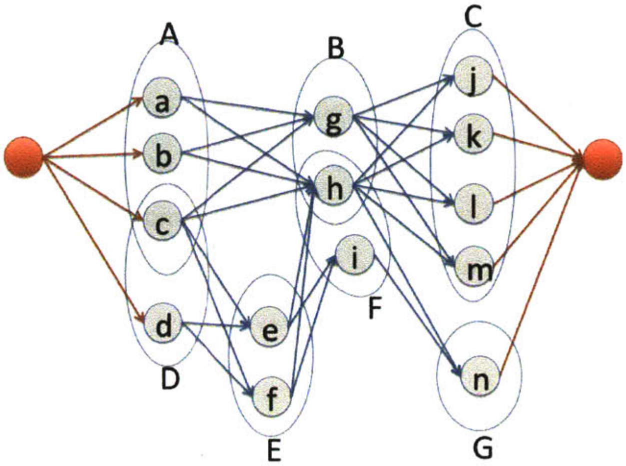


Figure 5-7: Constraint graph. The problem in Figure 5-3 can be transformed to a constraint graph shown in this Figure.

Basically, concatenating λ-optimal paths in SSPFNS is replaced by finding minimum-cost path on the constraint graph where edges are associated with λ-cost for the corresponding atomic constraints. As a matter of fact, the concatenation in SSPFNS can be understood as the minimum-cost path solution. This follows naturally from the fact that $SSPFNS \subset SSPGC$.

The SSPGC algorithm is described in Algorithm 11 using Algorithms 12 and 13 as building blocks. The algorithm first builds the constraint graph (Algorithm 12), then assigns the edge weight for the constraint graph by finding the λ-optimal path between a pair of subsequent visit nodes without any constraints, and finally it finds the minimum-cost path through the constraint graph (Algorithm 13).

Algorithm 11: Find-Optimal-Path-GC

```
Data: Graph and constraints
Result: optimal route satisfying the constraints
optimalPath := NULL;
constraints-graph = Build Constraint Graph(graph, constraints);
while ($\lambda$:=Find-Next-$\lambda$())$\neq$NULL do
    $\pi$ = Find-$\lambda$-Optimal-Path-GC($\lambda$, constraints-graph);
    optimalPath := Update-Regions-and-Optimal-Path($\pi$);
end
return optimalPath
```

Algorithm 12: Build-Constraints-Graph(λ)

```
Data: graph, constraints
Result: constraints-graph
for each pair of subsequent visit nodes $n_i$ and $n_{i+1}$ do
    Add edges of constraint to the atomicConstraintList;
end
From the atomicConstraintList build constraintsGraph;
```

Algorithm 13: Find-λ-Optimal-Path-GC(λ)

```
Data: $\lambda$, constraints-graph
Result: $\lambda$-optimal path satisfying the constraints
for each pair of subsequent visit nodes $n_i$ and $n_{i+1}$ do
    Find-$\lambda$-Optimal-Path($\lambda$, $n_i$, $n_{i+1}$);
end
Find the minimum-cost path from the constraint graph;
```

5.2.3 Correctness

Theorem 10. *The optimal path satisfying the general constraints occurs among the λ-optimal paths satisfying the constraints.*

Proof. Let point P on the mean-variance plane represent the optimal path satisfying the general constraints. Then, every other path point satisfying the constraints must be inside the parabola that passes through P having its apex at $(deadline, 0)$. Since the parabola is convex, P must be an extreme point for the convex hull including all the path points satisfying the constraints. □

Theorem 11. *The solution for Algorithm 13 is the λ-optimal path among the paths satisfying the constraints.*

Proof. We prove this by contradiction. Assume that there exists a path π that has smaller λ-cost than the solution of Algorithm 13, π^*_λ. Let a visit node sequence s^*_λ be an ordered sequence of visit nodes that π^*_λ satisfies. From Theorem 9, π^*_λ is the minimum-cost path among the paths that satisfy s^*_λ. Thus, π should follow another visit node sequence than s^*_λ. Since s^*_λ is the minimum-cost path for the constraint graph, any path that satisfies a visit sequence other than s^*_λ cannot have a smaller λ-cost than π^*_λ. Thus, the assumption is wrong. Consequently, no other path satisfying the constraints can have smaller λ-cost than the solution for Algorithm 13. □

5.2.4 Complexity Analysis

Let the number of nodes in the graph be N and the number of nodes in the constraint graph be M. Note that $M \leq N$ and $L = O(M^2)$, where L is the number of atomic constraints. First, Algorithm 12 takes $O(M^2)$ since there is a For loop that iterates for each of the atomic constraints. Now let us examine Algorithm 13. The For loop iterating through each atomic constraint uses the algorithm block 'Find-λ-Optimal-Path' described in Algorithm 2. The running time of the last line is $O(M^2)$. Thus, the running time of Algorithm 11 is $O(2 \cdot M^2 + L \cdot N^2 log^4 N)$. The For loop in Algorithm 13 can run in parallel since there is no computational dependency between finding λ-optimal paths for each atomic constraint. Depending on the available computation units, we can get the running time per computation unit based on the tradeoff between the running time and the number of computation units.

5.3 Extensions: Graph Transformation for Visit Cost and Dependency

5.3.1 Considering Visit Cost

We have assumed that the visit cost for visit nodes is 0. In this section we discuss how we can incorporate the visit cost into the algorithm. We assume that the visit cost follows a normal distribution $c_n \sim \mathcal{N}(m_n, v_n)$ independent from the visit cost distributions at other visit nodes and from the travel cost distribution at road segments. The constant visit cost $C(\geq 0)$ can be modeled with $m_v = C$ and $v_n = 0$.

We make a simple graph transformation of a visit node n into two nodes n' and n'' and a directed edge from n' to n'' associated with the visit cost. All the incoming edges to n are directed to n' and all the outgoing edges from n start from n'' in the transformed graph as in Figure 5-8. We can solve the problems with visit cost by running the developed algorithms on the transformed graph. This transformation increases the number of total nodes and edges by a factor of a constant less than or equal to 2. Thus, the complexity of the algorithms increases at most by a factor of 4 from the analysis in Section 5.2.4.

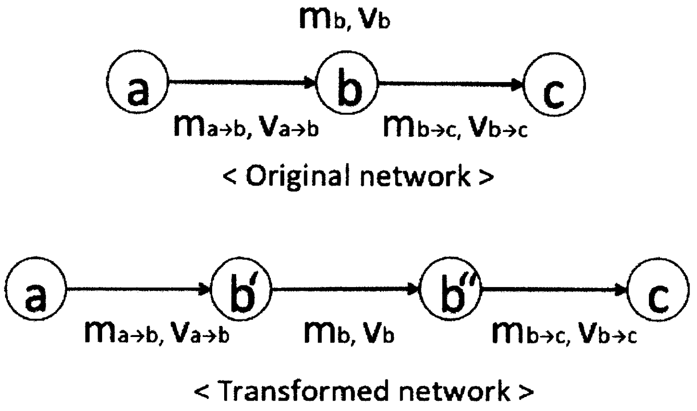
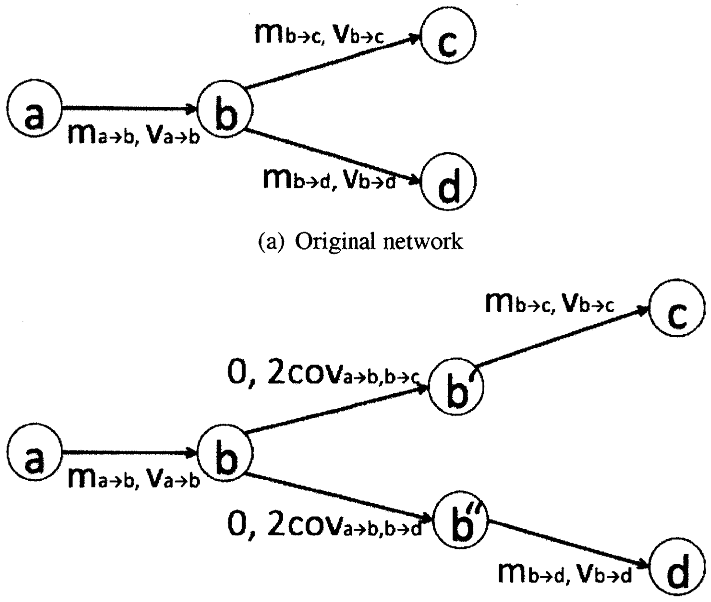


Figure 5-8: Graph transformation for visit cost. We transform the visit node b with mean m_b and variance v_b into two nodes b' and b'' and one edge connecting from b' to b''. All incoming edges to b are directed to b' and all outgoing edges from b start from b''. The edge weight of the new edge connecting b' and b'' is the same as the visit cost of b. $m_{b' \to b''} = m_b$ and $v_{b' \to b''} = v_b$.

5.3.2 Considering Stochastic Dependency

Although this chapter assumed the independence of each edge's distribution, correlations between adjacent edges can be easily considered by transforming the graph. Assume that there is a correlation between two adjacent edges $a \to b$ and $b \to c$. Then, $Var(w_{a\to b} + w_{b\to c}) = Var(w_{a\to b}) + Var(w_{b\to c}) + 2 \cdot Cov(w_{a\to b}, w_{b\to c})$. We introduce a new edge $b \to b'$ with mean 0 and variance $2 \cdot Cov(w_{a\to b}, w_{b\to c})$ as illustrated in Figure 5-9. This new edge captures the correlation between the two correlated distributions, maintaining the property that the variance of a path is the sum of the variances of all the edges in the path. The number of nodes and edges grows by one for each pair of correlation. Correlations between non-adjacent edges can be considered in a similar way. As long as we consider the correlations between an edge and a constant number of other edges, the complexity of our algorithms grow only by a constant factor. We can run our algorithms for the transformed graph and find the solution for the problems with correlation.



(a) Original network

(b) Transformed network

Figure 5-9: Graph transformation for dependency

5.4 Experiment

We implemented a Web-based route query system for the Singapore road network applying the algorithm developed in this chapter. The travel time statistics for each road segment were built from trajectories of a fleet of taxis with GPS receivers. The mean and variance for each road segment was saved for each one hour time slot for each day of the week. The user interface allows a user to input a) a time window when he wants to travel and b) the visit constraints by either typing in the address or selecting a location on the map interface and the deadline of the travel. Then, the system calculates the answer using the corresponding statistics and shows the path on the map.

Figure 5-10 shows an example of our system's path query result. We demonstrate that our algorithms significantly increase the probability of meeting a deadline by choosing the taxis that will reach the passenger in the minimum expected time. We also demonstrate that having multiple instances of agents and resources increases the probability significantly. The path query takes a few seconds with Python implementation on a Linux computer with 1GHz CPU and 1GB RAM. So, we can use this system for real-time applications such as

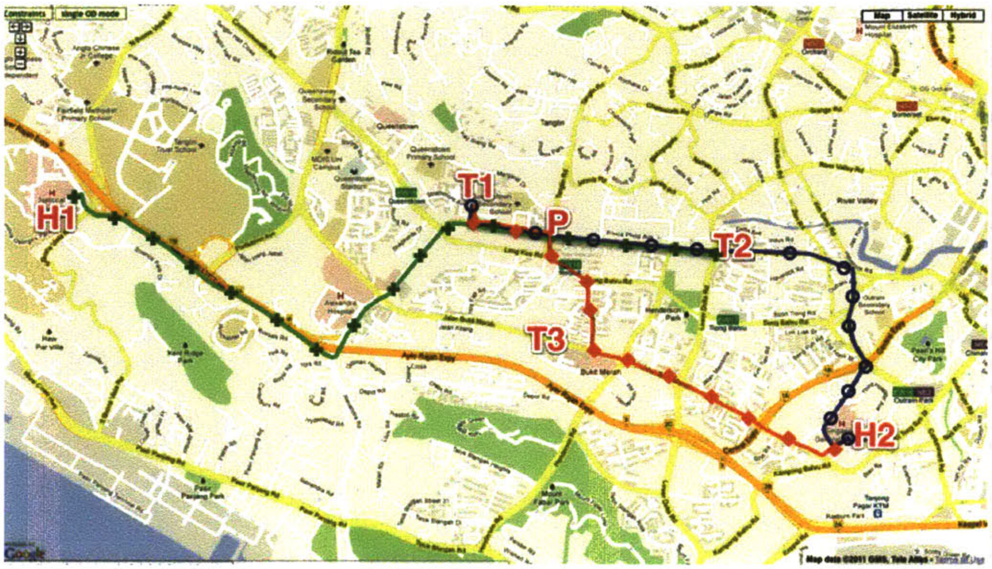


Figure 5-10: Hospital examples in Singapore: The patient located at P in the map wants to go to one of the two hospitals (indicated by H_1 and H_2) within 12 minutes in the early morning of a Monday. At the time, three taxis are available located at T_1, T_2, and T_3. The constraints can be represented as $(T_1 \vee T_2 \vee T_3) \wedge P \wedge (H_1 \vee H_2)$. The maximum probability path (blue line) is T_1 via P to H_2 with the probability 99 % of making the deadline. The expected travel time is 9 min 54 sec. Two other paths are shown for comparison. The shortest path (red line) is from T_1 via P to H_2 with probability 82 % and expected travel time 14 min 23 sec, and the minimum expected time path (green line) is from T_2 via P to H_1 with probability 94 % and expected travel time 8 min 28 sec.

taxi booking based on the taxis' current locations and the traffic conditions.

5.5 Synopsis

In this chapter, we defined the stochastic shortest-path problem with visit node constraints and developed efficient algorithms that solve the problem. The problem formulation captures an extensive range of important problems in motion planning for multiple agents with multiple locations for resources and destinations. For a given time budget or deadline, the results in this study provide the optimal choice of agents, resources, destinations, and the path to maximize the success probability. We demonstrated the algorithm by implementing a real-world navigation application based on the real data gathered by taxis in Singapore.

Chapter 6

Congestion-Aware Multi-Agent Route Planning

In the previous chapters, we developed a path planning algorithm for a group of agents considering uncertainty in travel time. In this chapter, we consider the congestion effect between agents: Each agent's path choice affects the travel time of other agents on the roads driven by the agent. This effect grows as we control a larger number of agents.

This chapter proposes a method for multi-agent path planning on a road network considering the congestion affected by each agent's path choice. We suggest a distributed method to find paths for multiple agents by introducing a probabilistic path choice achieving global goals such as the social optimum. This algorithm considers the travel delays affected by other agents' path choices. When applied in a large-scale, this algorithm is guaranteed to result in the minimum travel time for all the agents in the system.

This approach, which shows that the global goals can be achieved by local processing using only local information, can be parallelized and sped-up using massive parallel processing. We provide the analytical result on convergence and running time. We demonstrate and evaluate our algorithm by an implementation using asynchronous computation on multi-core computers.

This chapter is organized as follows: We formulate the planning problem in Section 6.1, and we develop a distributed algorithm for multi-agent routing in Section 6.2. In Section 6.3, we implemented the algorithm using a 64 core computer where each agent's task runs asynchronously on each core of the computer. We evaluate our algorithm using this implementation in a multi-core computer in Section 6.4.

6.1 Problem Set-up

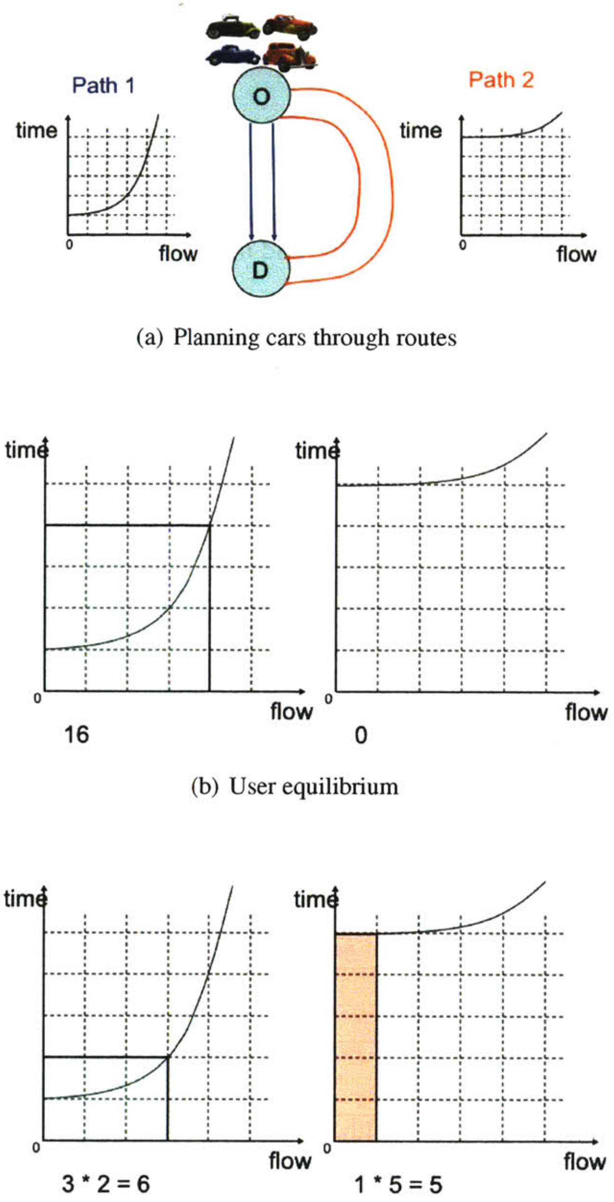
We are given a directed graph $G = (V, E)$ consisting of a set of vertices V that represent road intersections and a set of edges $E \subset V \times V$ that represent road segments between intersections. We are given origin and destination pairs for each agent $i \in A$. The goal is to plan paths for agents to achieve the social optimum, where each agent shares the edges of the given network with other agents.

To model the congestion generated by agents' path usage, we associate with edge e the travel time $t_e(f_e)$, which is a function of flow for e, f_e. Flow is defined as the number of agents per unit time. The link travel time $t_e(f_e)$ is assumed to be a differentiable non-decreasing function of link flow f_e.

Social optimum is achieved if and only if the agents' path plans minimize the sum of the total travel time for all the agents in the system. It is different from user equilibrium, which is achieved if and only if every agent's cost cannot decrease unless other agents change their decisions. User equilibrium is the state when selfish agents optimize their own paths. Social optimum may require some agents to take a sub-optimal path to minimize the total cost for the whole system. Figure 6-1 illustrates the difference between social optimum and user equilibrium.

Using a cost function that is agent-centric but captures neighborhood congestion, we would like for each agent to compute the path following a gradient decent type of method as illustrated in Figure 1-3. Each agent considers its origin and destination as well as those of its neighbors. Neighbors are agents that share some of their path with the current agent. At a conceptual level, our proposed solution uses a cost function that includes the path cost and the path congestion due to neighbors. By doing decentralized gradient descent, each agent will compute its own path that guarantees the system will reach the social optimum.

The solution algorithm has a few advantages. First, the algorithm is scalable since the computation is distributed for each agent and the communication is only required for



(a) Planning cars through routes

(b) User equilibrium

(c) Social optimum

Figure 6-1: Given the network with two road segments associated with the *cost vs. flow* relationships shown in (a), and given four cars traveling from *O* to *D*, equilibrium is achieved when every car takes Path 1 experiencing cost 4 as shown in (b). Social optimum is achieved when three cars take Path 1 experiencing cost 2, and one car takes Path 2 experiencing cost 5 as shown in (c). Whereas the total cost for user equilibrium is 16, that of social optimum is only 11.

neighbors. Second, the algorithm is reliable since the algorithm can still run even if an agent drops out from the system intentionally or with technical problems.

6.1.1 Notation

Throughout the chapter we use the symbols and notations given in Table 6.1.1.

Symbol	Description
G	a directed graph representing the road network
V	vertices representing road intersections
E	edges representing road segments
e	an edge
f_e	flow of edge e
t_e	travel time for edge e
A	a set of all the agents in the system
K_i	the number of paths for agent i
π_{ik}	agent i's k'th path
p_{ik}	probability that agent i selects his k'th path
c_{ik}	path cost of agent i's k'th path
$\underline{c_i}$	minimum of $c_{ik}, k = 1, \cdots, K_i$
τ	the time of the algorithm execution
V	the Lyapunov function that represents the total cost of the system
$V(\tau)$	V at time τ
V_i	the function that represents local cost for agent i
N_i	a set of neighbors for agent i including himself
V_{N_i}	sum of $V_j, \forall j \in N_i$
w_{ik}	the derivative of V_{N_i} by p_{ik}
y_{ik}	a processed version of w_{ik}
d_i	dependent path of agent i
$\mathbf{1}_K$	column vector of size K with all the elements being 1
e_l	element vector whose l'th element is one and other elements are all zero

6.2 Distributed Algorithm

In this section we describe a decentralized gradient descent algorithm for path selection. The algorithm is shown to converge to a social optimum using a Lyapunov proof.

The previous research on finding the user equilibrium for the road assignment problem showed the fact that the user equilibrium is achieved when the integral of the link delay is minimized [20]. But the method's convergence is not proven for a parallel implementation, and the convergence rate is not known for a serialized implementation. In our work, we consider a different Lyapunov-style cost function. The cost function we consider has two properties: (1) the cost function is represented as the sum of local cost function of each agent, and (2) the cost function converges to zero.

6.2.1 Probabilistic Path Choice

In this section, we provide a probabilistic path choice model. Each agent i is given a set of K_i origin to destination paths to select from. Let π_i be a K_i dimensional column vector representing agent i's K_i paths. We denote the k'th element of π_i as π_{ik} meaning the k'th path of agent i. We associate each path π_{ik} with the probability for agent i to take it among all the K_i paths. Similarly, we associate each path π_{ik} with the path cost c_{ik}. The cost for a path will be defined in the next section.

We assume that the number of agents in the system is large enough so that we can consider each agent's path choice probability as a fractional flow. We also assume that the congestion between agents is effective for the whole time horizon for planning. Thus, the flow of edge e can be expressed as the sum of the probability of each agent's path that includes edge e.

$$f_e = \sum_{\forall (i,k) \ s.t. \ e \in \pi_{ik}} p_{ik}$$

6.2.2 Local Cost Function

Let V_i be the cost function of agent i, which is a function of its own path choice probabilities and path costs. The total cost function of all the agents, V, can be described as

$$V = \sum_{\forall i \in A} V_i.$$

We set the local cost of agent i, V_i, for the social optimum as sum of cost difference between each path and the minimum cost path weighted by the flow.

$$V_i = \sum_{\forall k} p_{ik}(c_{ik} - \underline{c_i}), \tag{6.1}$$

where the path cost is defined as follows:[1].

$$c_{ik} = \sum_{\forall e \in \pi_{ik}} \left[t_e + f_e \frac{\partial t_e(f)}{\partial f} |_{f=f_e} \right], \tag{6.2}$$

and $\underline{c_i}$ is the minimum of path cost of all the paths, $\underline{c_i} = \min_{\forall k} c_{ik}$.

Note that V_i can be computed by agent i using only its information because its cost already takes into account the neighbors' cost via c_{ik}.

We define *marginal cost* of an edge as the total cost increase of all the agents using the edge by a small increase of the edge flow. As illustrated in Figure 6-2, social optimum occurs when all the paths' marginal costs are the same for all the agents. The reason we selected the cost of path as in Eq (6.2) is because c_{ik} becomes the same as the minimum of c_{ik}'s for every k with nonzero p_{ik} at the social optimum. Thus, each local cost is 0 at the social optimum as shown next.

Theorem 12. *$V_i = 0, \forall i \in A$ if and only if the flow is at the social optimum.*

Proof. if-part: Assume that the flow is at the social optimum. For any agent i, a path π_{ik} with nonzero p_{ik} has a path cost c_{ik} equal to the minimum path cost $\underline{c_i}$. Thus, for any agent i,

$$\begin{aligned} V_i &= \sum_{\forall k \ s.t. \ p_{ik}=0} p_{ik}(c_{ik} - \underline{c_i}) + \sum_{\forall k \ s.t. \ p_{ik} \neq 0} p_{ik}(c_{ik} - \underline{c_i}) \\ &= \sum_{\forall k \ s.t. \ p_{ik}=0} 0 \cdot (c_{ik} - \underline{c_i}) + \sum_{\forall k \ s.t. \ p_{ik} \neq 0} p_{ik} \cdot 0 \\ &= 0. \end{aligned}$$

[1] We can also find user equilibrium with the same algorithm developed in this chapter, when we replace the path cost with $c_{ik} = \sum_{\forall e \in \pi_{ik}} t_e(f_e)$

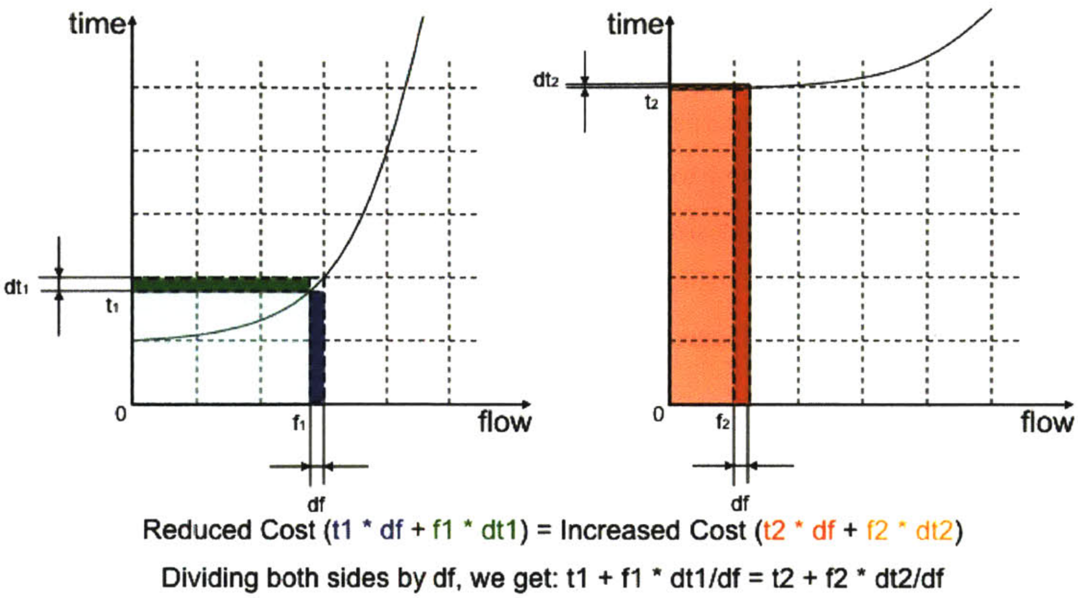

time
dt1
t1
0
f1
flow
df
time
dt2
t2
0
f2
flow
df
Reduced Cost (t1 * df + f1 * dt1) = Increased Cost (t2 * df + f2 * dt2)
Dividing both sides by df, we get: t1 + f1 * dt1/df = t2 + f2 * dt2/df

Figure 6-2: The condition that social optimum satisfies.

only-if-part: If $V_i = 0$,

$$\sum_{\forall k \ s.t. \ p_{ik}=0} p_{ik}(c_{ik} - \underline{c_i}) + \sum_{\forall k \ s.t. \ p_{ik}\neq 0} p_{ik}(c_{ik} - \underline{c_i}) = 0$$

Thus, all the paths with nonzero probability has the same cost as the minimum path cost for agent i. Since this holds for every agent i, the system is in social optimum. □

6.2.3 Distributed Control Law

We wish to develop a gradient controller that will drive the local cost to 0, in other words to the social optimum (Theorem 12). The distributed controller governs the time derivative of a agent's flow assignment, using the neighbors' current flow information. More precisely, the information needed by the controller is how much a neighbor's cost changes when the agent's path probability changes by a small amount. The controller moves its own path probability distribution among K_i paths into a newer distribution. The aggregate effect of these local controllers is proven to decrease the sum of the cost of all the agents. Moreover, we provide a proof that the rate of decrease is exponential in terms of the total cost.

Intuitively, each path's flow is controlled to move to (a) the negative direction of the normalized value of the neighbors' cost change derivative (b) multiplied by the agent's cost. (a) says that if the agent's path probability change affects the neighbors' cost increase, the agent's assignment of flow to that path should be decreased. (b) shows that the step a controller takes should be of a size proportional to the agent's current cost. This means that if the agent's cost is bigger than a neighbor's cost, the increase or decrease in the agent's flow control should have a bigger step.

The path probabilities for agent i, $p_{ik}, \forall k \in \{1, 2, \cdots, K_i\}$ satisfy the relation $\sum_{k=1}^{K_i} p_{ik} = 1$. We assign one variable as the dependent variable that compensates for changes in other variables. Thus, the $K_i - 1$ variables can change independently. For any small changes in the $K_i - 1$ variables, the constraint that the summation of the total probability should be 1 is held by this dependent variable. When we control an independent variable, we consider the change in the dependent variable.

Let N_i be a set of agent i and its neighbors. Let w_{ik} be the total cost increase by a small change in p_{ik}, defined as:

$$w_{ik} = \sum_{j \in N_i} \frac{\partial V_j}{\partial p_{ik}}$$

where

$$\frac{\partial V_j}{\partial p_{ik}} = \frac{\partial}{\partial p_{ik}} \sum_{\forall l, \pi_{jl}} p_{jl}(c_{jl} - \underline{c_j})$$

where

$$\frac{\partial c_{jl}}{\partial p_{ik}} = \sum_{\forall e \in \pi_{ik} \& \pi_{jl}} \frac{\partial c_e}{\partial p_e} - \sum_{\forall e \in \pi_{id} \& \pi_{jl}} \frac{\partial c_e}{\partial p_e}, \ \forall i, j \in A$$

where A is the set of all the agents. Note that when we increase or decrease the control variable, we decrease or increase the dependent variable by the same amount, respectively.

Let us define y_i as follows for $k \neq d_i$:

$$y_{ik} = \begin{cases} 0, & \text{if } p_{ik} = 0 \text{ and } w_{ik} > 0 \text{ or } p_{ik} = 1 \text{ and } w_{ik} < 0 \\ w_{ik}, & \text{otherwise.} \end{cases} \tag{6.3}$$

Let d_i be the dependent path of agent i. Let $d_i = \text{argmax}_{\forall k}\, p_{ik}$. $p_{id_i} = 1 - \sum_{\forall k \neq d_i} p_{ik}$. Our distributed controller is described as follows:

$$\frac{\mathrm{d}p_{ik}}{\mathrm{d}\tau} = \begin{cases} -\gamma V_i \frac{y_{ik}}{\|y_i\|^2} & \text{for } k \neq d_i \\ -\sum_{k \neq d_i} \frac{\mathrm{d}p_{ik}}{\mathrm{d}\tau} & \text{for } k = d_i, \end{cases}$$

where γ is the step size, which can be chosen by the controller designer. We discuss the selection of the γ value in Section 6.3.

Combining the equations above we get:

$$\boxed{\text{Control Law: } \frac{\mathrm{d}p_i}{\mathrm{d}\tau} = -\gamma V_i \frac{y_i - (\mathbf{1}_{K_i}{}^T y_i) e_{d_i}}{\|y_i\|^2}} \tag{6.4}$$

where $\mathbf{1}_{K_i}$ is a column vector of size K_i with all the elements set to 1, and e_{d_i} is a column vector of size K_i with the d_i'th elements set to 1 and the rest set to 0.

Equation (6.3) determines the control variables satisfying the boundary conditions. Since the $\mathbf{1}_{K_i}{}^T y_i = \sum_{k=1}^{K_i} y_{ik}$, $(\mathbf{1}_{K_i}{}^T y_i) e_{d_i}$ in Equation (6.4) sets the dependent variable as the negative of the sum of all the control variables.

```
Algorithm 14: Distributed algorithm for agent $i$
Data: Origin, Destination, $K_i$: the number of paths, $\varepsilon$: tolerance of solution
Result: Path probability of each path
Initialization;
Find $K_i$ minimum expected time paths based on the historical data;
Find neighbors that share any road segment with this agent;
while $V > \varepsilon$ do
    Get $\frac{\partial V_j}{\partial p_i}$ from neighbors $j \in N_i$ ;
    Find path probability using the control law in Equation (6.4);
```

6.2.4 Convergence Analysis

Theorem 13. *The controller in Equation (6.4) makes the total cost function $V(\tau)$ decrease exponentially.*

Proof. First, we find the time derivative of V with respect to τ.

$$\frac{dV}{d\tau} = \sum_{\forall i \in A} \left(\frac{\partial V}{\partial p_i}\right)^T \frac{dp_i}{d\tau} = \sum_{\forall i \in A} \left(\frac{\partial V}{\partial p_i}\right)^T \frac{dp_i}{d\tau}$$

Since $V = \sum_{\forall j \in A} V_j$,

$$\frac{dV}{d\tau} = \sum_{\forall i \in A} \left(\frac{\partial}{\partial p_i}\left[\sum_{\forall j \in A} V_j\right]\right)^T \frac{dp_i}{d\tau} = \sum_{\forall i \in A} \left(\sum_{\forall j \in A} \frac{\partial V_j}{\partial p_i}\right)^T \frac{dp_i}{d\tau}$$

Since the partial derivative by p_j only affects j's neighbors,

$$\frac{dV}{d\tau} = \sum_{\forall i \in A} \left(\sum_{j \in N_i} \frac{\partial V_j}{\partial p_i}\right)^T \frac{dp_i}{d\tau}$$

Plugging in our controller Equation (6.4), we get

$$\begin{aligned}\frac{dV}{d\tau} &= \sum_{\forall i \in A} \left(\sum_{j \in N_i} \frac{\partial V_j}{\partial p_i}\right)^T \left(-\gamma V_i \frac{y_i - (\mathbf{1}_{K_i}{}^T y_i) e_{d_i}}{\|y_i\|^2}\right) \\ &= \sum_{\forall i \in A} -\gamma V_i \left(\sum_{j \in N_i} \frac{\partial V_j}{\partial p_i}\right)^T \left(\frac{y_i - (\mathbf{1}_{K_i}{}^T y_i) e_{d_i}}{\|y_i\|^2}\right)\end{aligned}$$

Since the nonzero elements of vector y_i except for the d_i'th element are equal to those of $\sum_{j \in N_i} \frac{\partial V_j}{\partial p_i}$, and since the d_i'th element of $\sum_{j \in N_i} \frac{\partial V_j}{\partial p_i}$ is 0,

$$\frac{dV}{d\tau} = \sum_{\forall i \in A} -\gamma V_i \left(\frac{y_i{}^T y_i}{\|y_i\|^2}\right) = -\gamma \sum_{\forall i \in A} V_i = -\gamma V. \qquad (6.5)$$

Integrating both sides of Eq 6.5 in terms of τ, we get:

$$V(\tau) = V(0)e^{-\gamma\tau}.$$

Thus, the total cost function decreases exponentially. □

6.2.5 Running Time

In the following theorem, we quantify the time needed to converge to an arbitrarily small range of the optimal solution.

Theorem 14. *For any $\varepsilon > 0$, we have $V(\tau) < \varepsilon$ for $\tau > \frac{1}{\gamma}log(\frac{V(0)}{\varepsilon})$.*

Proof. Since $\tau > \frac{1}{\gamma}log(\frac{V(0)}{\varepsilon})$, we have

$$\begin{aligned} V(\tau) &= V(0)e^{-\gamma\tau} \\ &< V(0)e^{-\gamma\{\frac{1}{\gamma}log(\frac{V(0)}{\varepsilon})\}} = \varepsilon. \end{aligned}$$

□

Assuming the polynomial relationship between edge travel time and edge flow, we find the running time needed to converge to an arbitrarily small range of the optimal solution in terms of the number of agents in the system.

Theorem 15. *Let N be the number of agents in the system. If the congestion model $t_e(f_e)$ is given by a polynomial function of f_e with order m, $V(\tau) < \varepsilon$ for $\tau > \frac{m}{\gamma}O(log(\frac{N}{\varepsilon^{\frac{1}{m}}}))$, for any $\varepsilon > 0$.*

Proof. The worst case happens when all the agents share the same edge. Thus, the complexity of the initial cost V(0) can be described as $O(N^m)$. Plugging this into the statement in Theorem 14, we get $\tau > \frac{1}{\gamma}O(log(\frac{N^m}{\varepsilon})) = \frac{m}{\gamma}O(log(\frac{N}{\varepsilon^{\frac{1}{m}}}))$. □

6.3 Asynchronous Implementation

We implemented the algorithm in C++. A C++ thread was created for each agent. One thread is designated to periodically check whether the cost function V reached the predefined error tolerance ε. This thread gets the local cost V_i from each thread and sums up to get V, and if it detects $V < \varepsilon$ it finishes the program. The communication between threads was implemented using shared memory and Mutex. The threads run asynchronously. As long as there are enough cores, different processors are assigned to different thread.

We can rewrite Equation (6.4), as follows:

$$\mathrm{d}p_i = -\gamma V_i \frac{y_i + (\mathbf{1}_{K_i}{}^T y_i) e_{d_i}}{\|y_i\|^2} \mathrm{d}\tau.$$

$\mathrm{d}\tau$ is determined by the processing time of one iteration of each processor. Different processors may have different $\mathrm{d}\tau$ for they may have different computing powers and also they may need different amount of computation. Our algorithm works in these kinds of varieties with a fully asynchronous implementation. Each processor runs Algorithm 14 asynchronously.

The step size γ affects the running time, thus we need to choose a good γ by doing experiments. The right γ seems to be dependent on the problem size. When the processing time of one iteration is denoted as T, a reasonable and good choice of γ is $\gamma = \frac{Constant}{T}$ since this maintains the step size of one iteration to be constant. Since the processing time of one iteration depends on the number of neighbors, this choice of γ provides good scaling as $O(N\ logN)$ according to Theorem 15 for a fixed m and ε.

6.4 Evaluation

We ran the C++ implementation on a Tilera chip with 64 CPU cores [5].

6.4.1 Exponential Convergence

Figure 6-3 shows an exponential decrease of V for 50 agents in a Boston road network.

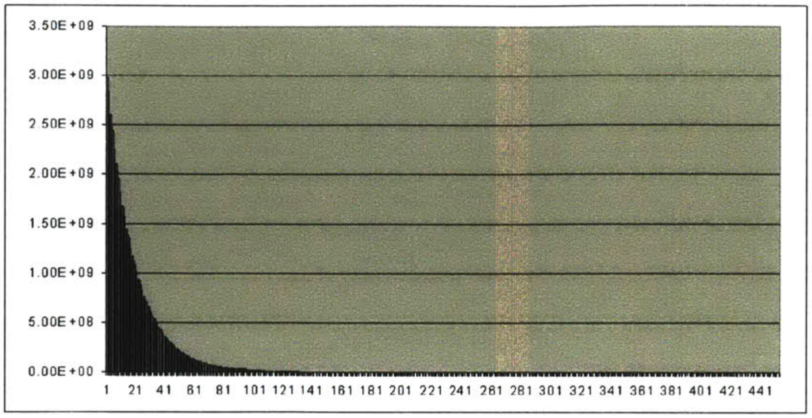


Figure 6-3: Exponential convergence of our cost function V. Run on 64 core computer. x-axis: seconds, y-axis: cost function V

6.4.2 Scalability

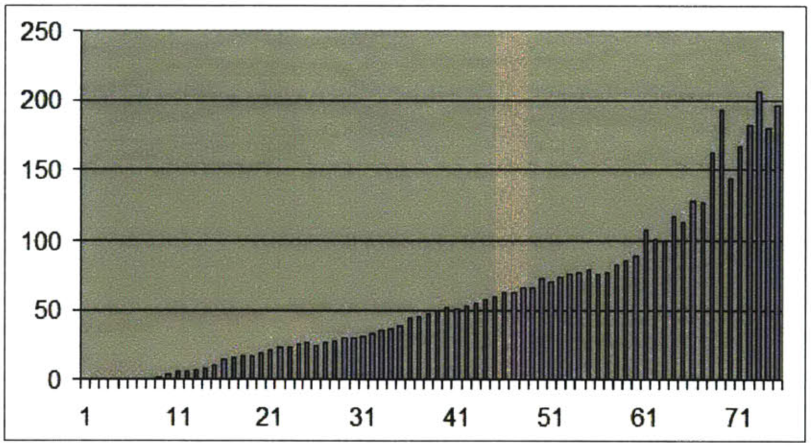
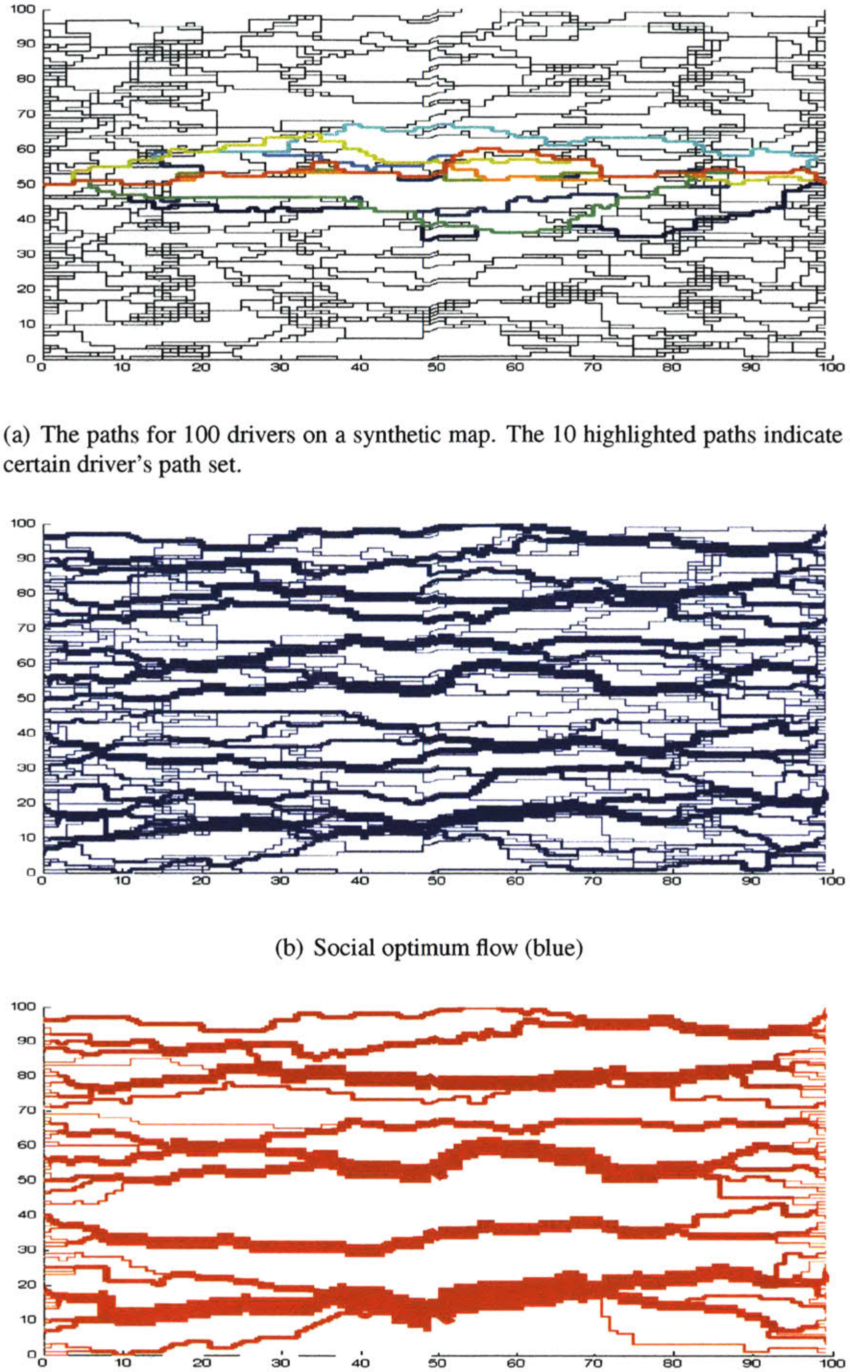


Figure 6-4: Running Time in 64 cores computer. x-axis: number of agents, y-axis: time in seconds. The scaling of our algorithm appears to be $O(nlogn)$.

Figure 6-4 shows our algorithm scales as analyzed in the previous section, which is $O(NlogN)$ when N is the number of agents. We can see sudden increase in the running time in Figure 6-4 around the 63rd agent. This is because we have only 63 available cores for agent threads. Our algorithm scales well for large number of CPUs getting full benefit from the number of CPUs.



(a) The paths for 100 drivers on a synthetic map. The 10 highlighted paths indicate a certain driver's path set.

(b) Social optimum flow (blue)

(c) Greedy flow (red)

Figure 6-5: Grid network

6.4.3 Implementation in Grid Network

On a grid network with 100×100 nodes and bidirectional edges associated with the congestion cost-flow relationship, given 100 agents' origin and destination and ten paths per agent, the algorithm finds the social optimum. Edge cost was generated with $t_e = T_0(1 + \alpha(\frac{f_e}{C})^\beta)$, where T_0 is selected randomly from 0 to 1, and $C = 10$, $\alpha = 0.15$, $\beta = 4$. Agent i's origin is the i'th leftmost point from the bottom and its destination is the i'th rightmost point from the bottom in Figure 6-5(a). In Figures 6-5(b) and 6-5(c) the thickness of the path indicates the flow of social optimum (blue) and greedy assignment (red). We can observe that the social optimum flow distributes agents through more road segments than greedy assignment, whereas greedy assignment uses fewer road segments that would have been good without congestion. The total cost of the system is 3720 for the social optimum and it is 4550 for the greedy assignment.

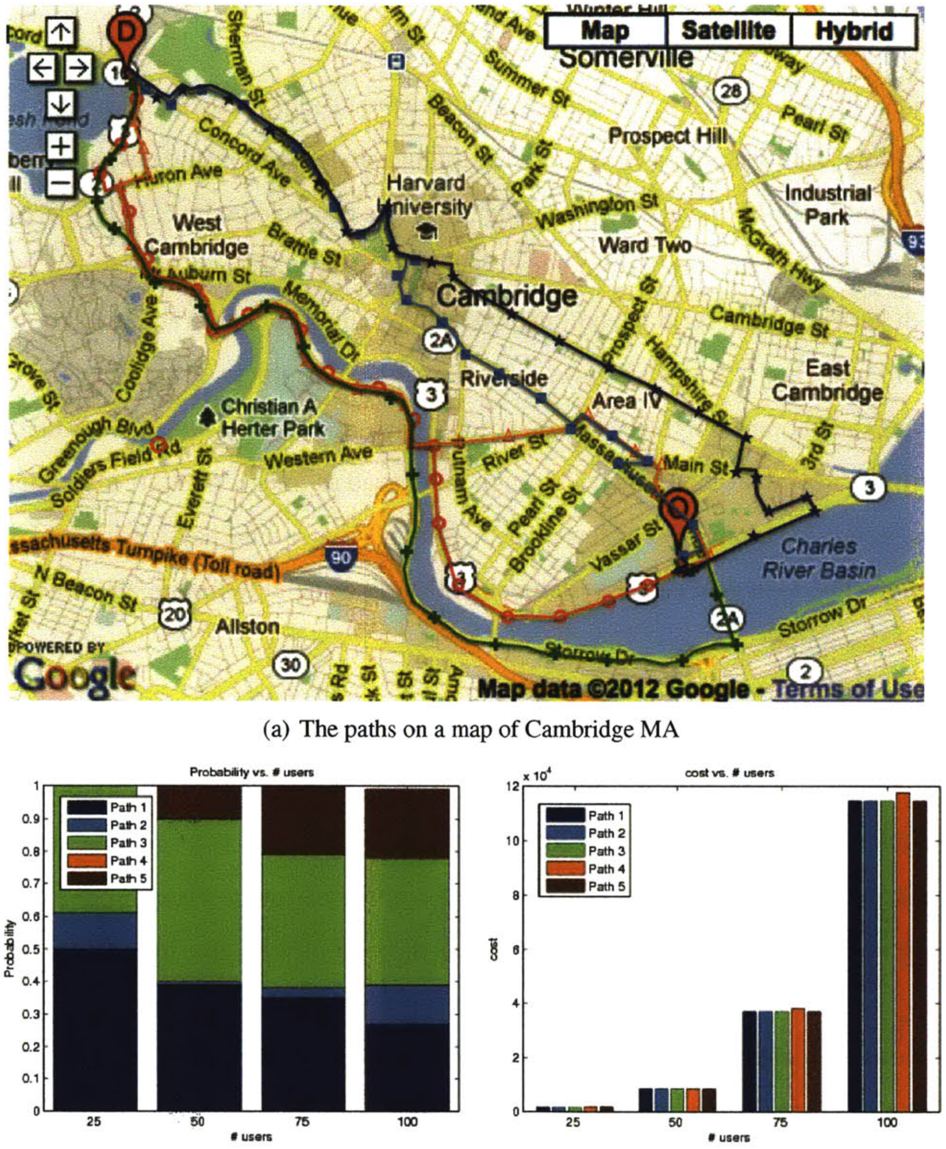
6.4.4 Path Example in Real Road Network

We picked 5 paths from MIT (denoted as "O" in the map) to Fresh Pond (denoted as "D" in the map). We ran our algorithm for different number of agents planning to achieve the social optimum. In Figure 6-6, we can see that the cost of the paths are the same for the paths that have non-zero probability. We can observe that the portion of people who choose longer paths increase as the number of agents grows. We can observe that Path 4 was never used since its cost is always greater than the others.

6.4.5 Discussion

Comparison with Greedy Algorithm

Figure 6-7 shows the difference in total time between our algorithm and a greedy algorithm. The greedy algorithm assigns the best path for each agent when each agent's path is determined using the information of the previously planned agents. Thus, the quality of the solution depends on the order of planning agents. Figure 6-7 shows that the difference is significant and increases as large numbers of agents plan together.



(a) The paths on a map of Cambridge MA

(b) The probability of each path for a driver

(c) the cost of each path for a driver

Figure 6-6: On a real network in Boston, we illustrate the social optimum solution for 25, 50, 75, and 100 drivers leaving 'O' and arriving at 'D', respectively. (b) shows the probability of choosing one of the five paths depending on the number of total drivers. The result shows that Path 5 should not be used at all when the number of agents is small, but it needs to be used significantly for the case when more agents participate. (b) shows the cost c_{ij} is the same for every path used with non-negative probability.

Effect of Probabilistic Assignment

Figure 6-7 also shows how the solution is different when we actually assign one path for each agent according to the probability of path choice. The realization of the result probability distribution is denoted as ‘Alg. 1 atomic’ in Figure 6-7. We can see that the atomic realization is not very different from the fractional assignment with the probability.

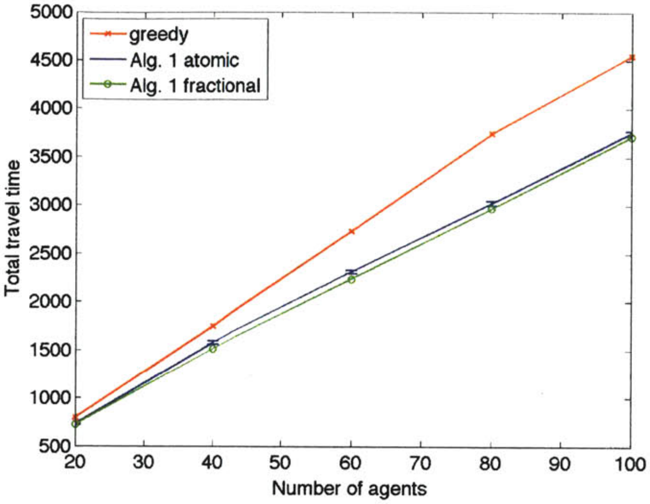


Figure 6-7: The plot shows the the comparison with greedy algorithm and the effect of realization to atomic flow based on the probability result. ’Alg. 1 atomic’ shows the average of 50 sets of runs with error bars indicating the standard deviation.

Choice of Number of Paths

Figure 6-8 shows how the solution quality varies if we take more paths into consideration. We observe that for a small number of agents more paths do not help much. We can observe that for a larger number of agents we can observe some decrease of the total travel time when we use more paths. However, the decrease saturates at about 15 paths. This fact indicates that considering about 15 paths is enough in real road network. We plan to extend our algorithm to searching the path with the smallest path cost at each iteration and including it to the path set. By doing this, we can guarantee that the found path is the optimal path among all the possible paths from origin to destination with the additional computational cost of searching shortest path.

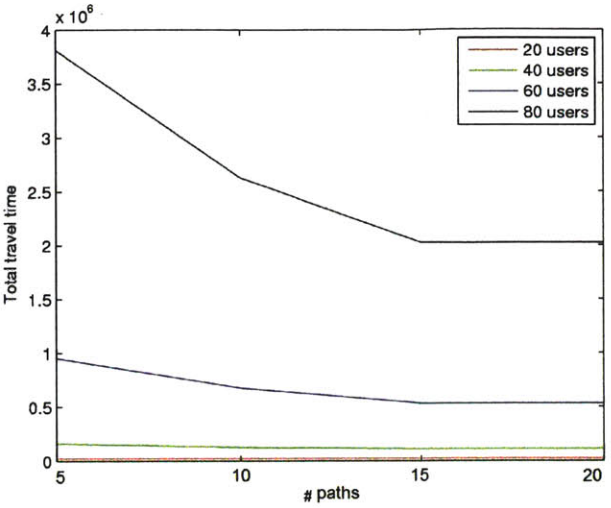


Figure 6-8: Total travel time vs. different number of paths.

6.5 Synopsis

In this chapter, we developed a distributed algorithm for multi-agent routing. The algorithm first finds the optimal fractional flow viewing the drivers' decisions as fractional flows. We proved that the local processing based on the local information achieves social optimum with exponentially fast convergence. We used the fractional flows as the probability of selecting among the predefined path set. We showed that the probabilistic assignment for atomic drivers generates very close travel time to that of the fractional flow case. We implemented our algorithm in an asynchronous way and tested it on multi-core computers where each processor runs for each agent's task.

Chapter 7

From Theory to Practice: Traffic Routing System with Sensor Data

In Chapter 6 we introduced a decentralized, theoretically-proven algorithm for multiple agents to navigate through a road network minimizing the total travel time of the agents. In this chapter, we present a congestion-aware route planning system that implements the multi-agent routing algorithms, and we describe the details of the data we use for our route planning system. The parameters of the models that the algorithms use are estimated from the real sensor data. We use two kinds of traffic sensor data: One is the GPS trace data from probe vehicles. The other is the data from inductive loop detectors. The loop detector data available in limited locations serve as valuable data for estimating flows in conjunction with the taxi data. We provide evidence that the taxis move randomly so that they cover almost the whole area.

Congestion characteristics of each road segment is required for our congestion-aware routing system. The detailed methods for traffic flow estimation are presented in Chapter 8. The methods for detecting congestion for all the road segments are provided in Chapter 9. Chapter 10 provides our technique for estimating congestion function based on estimated traffic flow and presents our experimental results using real trip data from Singapore taxis.

The organization of this chapter is as follows: Section 7.1 describes the route planning system for multi-user congestion-aware routing. The system uses the traffic planning algorithm provided in Chapter 6 for finding the socially optimal assignment of traffic. Section 7.2 describes our data for inferring traffic speed and flow. Section 7.3 provides an intuition for why taxis make for good dynamic probes, by showing that they have a rapidly mixing property.

7.1 System

We present a congestion-aware route planning system. The system is composed of database server and Web server. The origin, destination, deadline, and the departure time of trips are provided as input through a front-end web interface provided by the system. The back-end computational module computes the social optimum paths using the algorithm described in Chapter 6. That algorithm finds the congestion-aware social optimum paths for all the drivers in the system. The algorithm decides which path among a given set of paths for each driver should be selected by an iterative computation process. The algorithm is decentralized and scales well for very big problems with many agents and a large-scale network of roads. The front-end web interface is implemented in Javascript. We use Google maps open API [2] for visualizing paths and map overlays. The back-end processing module is implemented in C++.

7.2 Data

For the traffic models we use, we need either traffic delay or traffic flow information, or both. To infer this information, in this study we use two sources of data: taxi data from a large fleet of taxis in Singapore and loop detector data in Singapore. The taxi data provides the speed and flow information for all the road segments. The taxi speed data is used as ground truth on traffic speed. The loop detector count data is used as ground truth on traffic volume. Whereas the loop detector count data is only available in a non-real-time manner for limited locations, the taxi count data is available for almost all the road segments in the road network in real time. However, the taxi count data is only a sampled version of the total traffic count. Table 7.1 summarizes the availability of the data in location and time as well as the capability for getting speed and flow information.

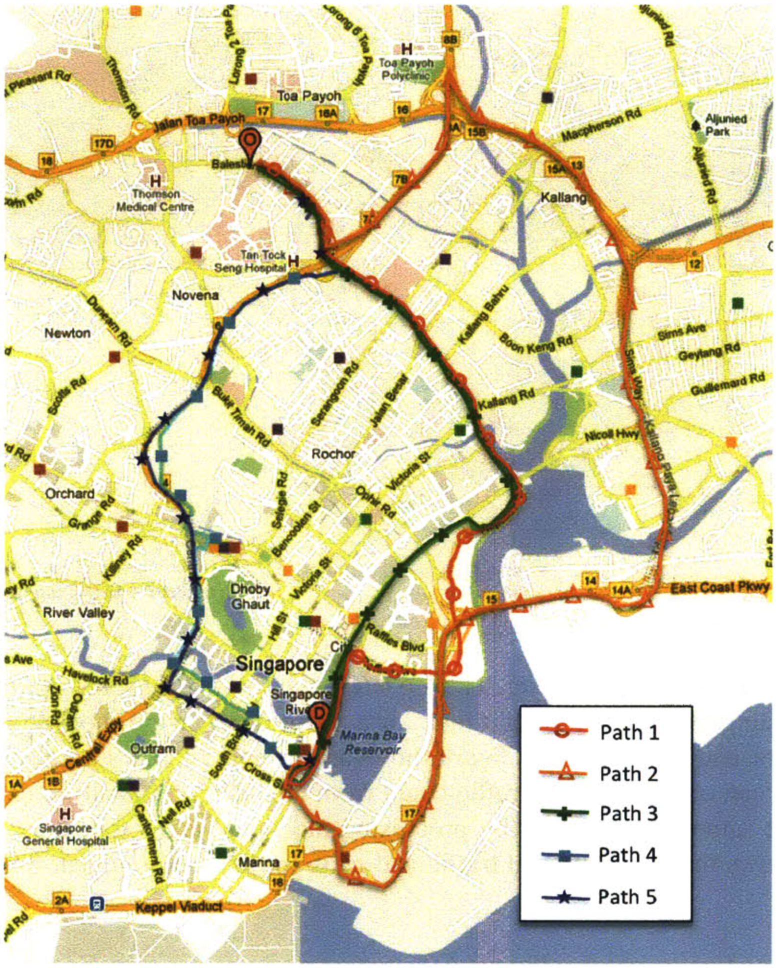
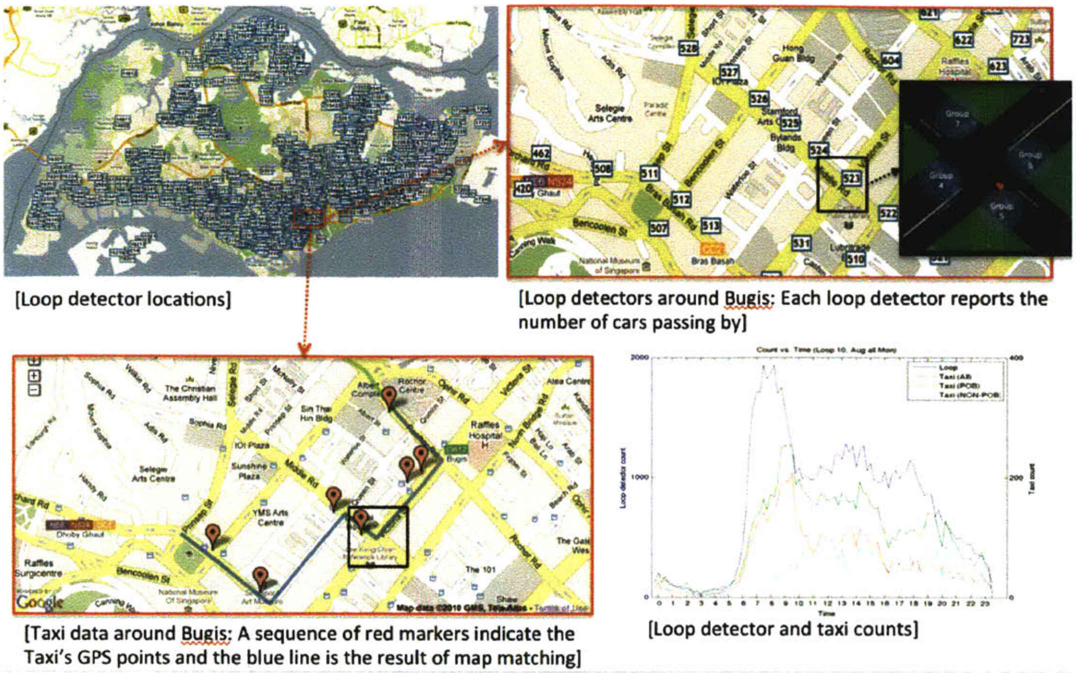


Figure 7-1: A snapshot of the route planning system's web interface for Singapore. Among the 5 paths indicated on the map, path 1 is the path that a taxi took at 8 am on Monday, August 2, 2010 starting from the location marked 'O' and ending at 'D'. The system computed the green-colored path as the best path for achieving the social optimum.

Table 7.1: Taxi data and inductive loop detector data

Data Source	Location Availability	Time Availability	Speed	Flow
Taxi	Almost everywhere	Real time	Ground truth	Sampled count
Loop detector	Limited locations	Historical	Not available	Ground truth

As shown in Table 7.1 taxi is a good source for real-time speed information for almost every road segment. Section 7.3 provides the evidence of the taxis' rapid movement throughout the entire area of the road network. In addition, loop detector data is a good



[Loop detector locations]

[Loop detectors around Bugis: Each loop detector reports the number of cars passing by]

[Taxi data around Bugis: A sequence of red markers indicate the Taxi's GPS points and the blue line is the result of map matching]

[Loop detector and taxi counts]

Figure 7-2: Loop data and taxi data. The top left figure shows the location of inductive loop detectors in Singapore. The top right figure is a zoomed-in version showing the individual loop detectors, and the bottom left figure shows an example taxi GPS trajectory. For a road segment, we can obtain the total traffic count from loop detectors, and we can get the taxi count for the same road segment by processing individual taxi's trajectory. The bottom right plot shows the count of total traffic and taxis for the road segment during a certain day.

source for flow information. However, the information is only available in a non-real-time manner and the availability of the data is limited to the location where the loop detectors exist. In Chapter 8, we show how to estimate the real-time general traffic flow for all the road segments. We show that we can estimate the general traffic flow accurately using loop detector data as well as taxi data.

7.2.1 Loop Detectors

Our study also uses the the loop count data we obtained from the Land Transportation Authority (LTA) in Singapore for 12,500 loop detectors in 1,300 intersections in Singapore for the same period of time, August 2010. Each loop detector record gives the number of cars that pass over each loop detector during a 15-minute time slot. There are 2,688 time-slots for the first four weeks of August 2010. We used these time slots for our studies and map all the taxi and loop detector data into these slots. The loop count serves as the ground truth for the general traffic.

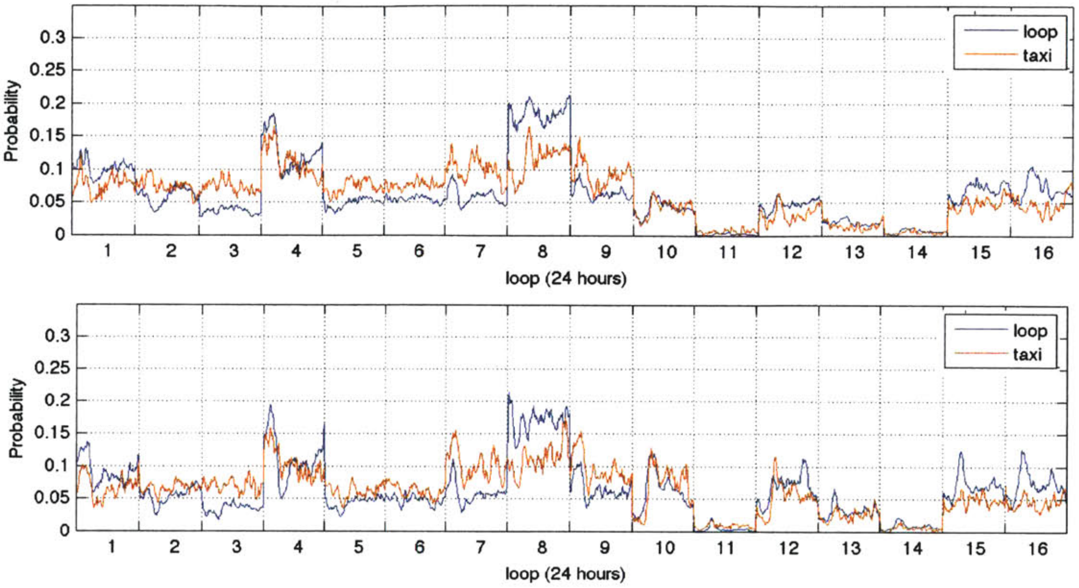


Figure 7-3: Distribution for August 1 and August 2, 2010 over 16 selected road segments. The x-axis includes 16 segments, one per road. Each of these 16 segments includes 96 points, one for each 15-minute time slot during a 24-hour day. The y-axis shows the fraction of traffic at that location and time.

Figure 7-3 shows the taxi and loop detector count data for 16 Singapore road segments we selected randomly. Note that the taxi distribution (in red) tends to overestimate the loop distribution (in blue) during much of the day, and that the overestimation varies. During the morning rush hour, the taxi and loop distribution values are nearly identical. Thus, while there exists a bias, this bias certainly changes throughout the day, though it appears relatively consistent across days.

7.2.2 Probe Vehicles

Our study uses four weeks of data (August 2010) from 16,000 taxis in Singapore, which amounts to approximately 500 million data points (31GB). Each taxi record contains the car id, the driver id, the time stamp, the latitude, the longitude, and state of operation (represented by one of the following four attributes: free, person on board, busy, on break). Records are logged at intervals between 30 seconds and 2 minutes, depending on network connectivity. Table 7.2 shows the raw GPS data from the probe vehicles.

Table 7.2: Format of taxi GPS data

Field	Description
Time	Vehicle reported date in the format dd/mm/yyyy hh:mm:ss
Vehicle ID	identification number for vehicle
Driver ID	identification number for driver
Longitude	X co-ordinate (longitude) of the location reported by vehicle
Latitude	Y co-ordinate (latitude) of the location reported by vehicle
Speed	Speed of the vehicle in km per hour at the reported time
State	State of vehicle. Following are the state: FREE - Vehicle is free to take a passenger POB - Passenger is on board BUSY - Vehicle is not free to take a passenger BREAK - Vehicle is not free to take a passenger

Map Matching

Processing the taxi data to get the counts and speeds requires several steps. First, the data is mapped to a time series of GPS points for each car. Next, we match the time series of GPS points to a sequence of road segments in the road network of Singapore. To overcome the noise and sparsity of the taxi GPS data, we used a map matching method based on the Viterbi algorithm, as in [99]. Third, we count the number of taxis on road segments. From this process, we get the taxi counts for each location in the road, which is regarded as the sampled count for the probe traffic. Fourth, we calculate the speed of each taxi when it passes each road segment. This speed, calculated from the taxi movement, is regarded as a ground truth speed for the road segment.

7.3 Rapid Coverage by Probe Vehicles

Our thesis is that no matter where an individual taxi is located, or where it starts its day, its location will rapidly become indistinguishable from the overall taxi distribution. The reason for this is the stochastic nature of a taxi's passengers—unlike a mail delivery truck that likely follows a fixed route, a taxi will randomly visit locations based on the random nature of the passengers' destinations.

In effect, one can view a taxi's movement as a random walk, with transitions between regions governed by the conditional probabilities of passenger movements from origins to destinations. Although the driver does maintain full control of the taxi's location when there are no passengers on board, we show empirically that taxi distributions tend to converge regardless of the taxis' initial locations. Thus, the locations of the taxis can be viewed as having been randomly drawn from the overall taxi distribution at any point in time. As such, they are well suited to the problem at hand: Regardless of the initial locations of the dynamic taxi probes, we can expect them to quickly spread, providing good coverage in all regions.

In the remainder of this section, we analyze taxi movement over 27 regions (see Section 7.3, less one outlier region) at 15-minute intervals. The transition probabilities are empirically derived from the taxi data.

Partitioning of Singapore into Traffic Zones

To make the analysis more tractable, we partitioned the area of Singapore into a number of regions. This was done in a completely data-driven fashion, by applying the standard K-means clustering on a subset of the origins and destinations extracted from the taxi data. (198721 origins and destinations from trips of 500 taxis over the first week were used for the K means clustering.) The K-means clustering algorithm iteratively assigns each origin or destination to one of K centroids to which the origin or destination is closest (as measured by Euclidean distance), and then adjusts each of the K centroids to the means of the origins and destinations assigned to it. This results in a partition of the data space into Voronoi regions, each represented by one of the K means or centroids.

In choosing the value of K, we followed [93] in the use of a criterion, composed of a linear combination of the *distortion* (or encoding error) and a regularization term:

$$\sum_{i=1}^{N}(\vec{x}_i - \vec{c}_{\mathrm{ENCODE}(\vec{x}_i)})^2 + 2\lambda K \log N$$

where N is the number of origins and destinations, $\vec{x}_i$ are the GPS locations of the origins and destinations, $\vec{c}_{\text{ENCODE}(\vec{x}_i)}$ is the nearest centroid to $\vec{x}_i$, and λ is the regularization factor. In this criterion, the distortion $\sum_{i=1}^{N}(\vec{x}_i - \vec{c}_{\text{ENCODE}(\vec{x}_i)})^2$ is the sum of Euclidean distances of each origin or destination from its nearest centroid, and $2\lambda K \log N$ is the regularization term. The "optimal" K^* is chosen to minimize this criterion.

For our analysis, we have chosen $\lambda = 35$, which results in $K^* = 28$. Although the "optimal" K is sensitive to the choice of λ, we believe that our choice is validated for the following reasons:

1. By visually examining the graph of distortion against K, we find an "elbow" at around $15 \leq K \leq 35$;

2. There happen to be 28 postal districts in Singapore [104];

3. The resultant Voronoi regions appear to correspond to semantically significant areas in Singapore.

We also verified that observations presented in this thesis are robust to a wide range of $K = 15, \ldots, 35$.

Figure 7-4 shows the Voronoi regions extracted through K means clustering (27 regions are shown, with an outlier region excluded). Each dot in the plot represents a single origin or destination of a taxi trip, and is color-coded to represent the region to which it belongs. Hence, each region is represented by a continuous patch of color-coded dots. Although no additional map information was used in the creation of the plot, the map of Singapore clearly emerges from the data, with the empty patches corresponding to inaccessible areas (e.g., reservoirs or forests). We observe that semantically significant regions such as the Airport in the East, the Central Business District and Orchard regions in the South; Tuas in the West, and Woodlands in the North, are also visible.

Taxis Are Rapidly Mixing

To quantify the rate of convergence, we compute the maximum difference (i.e., the L^∞ distance[1]) between any initial taxi distribution and the observed overall taxi distribution

[1] A low L^∞ distance indicates that the L^1 and L^2 distances must necessarily be low as well.

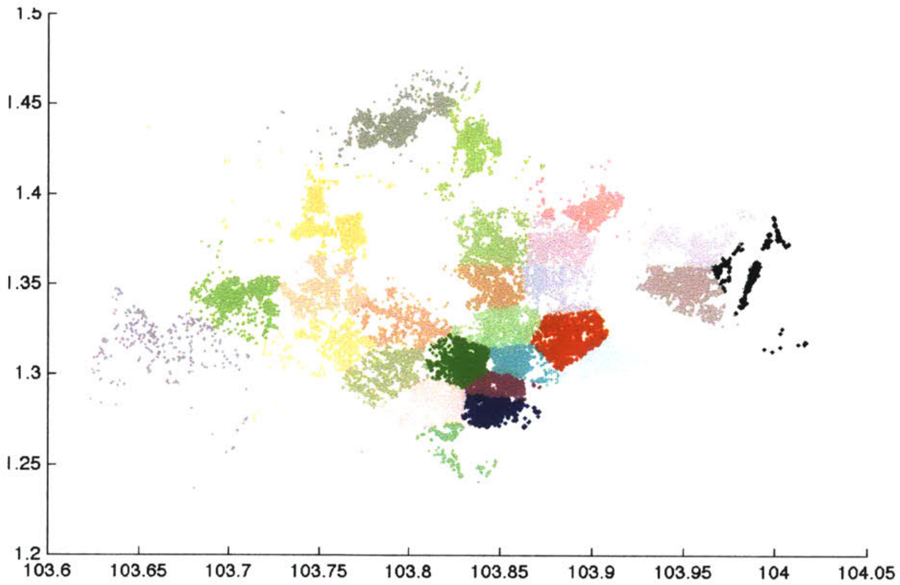


Figure 7-4: Voronoi regions of Singapore, extracted through K means clustering on origins and destinations of taxi trips. A total of 27 regions are shown in this map, with one outlier region excluded. The highlighted 6 regions will be discussed in detail in Section 10.1.3.

after 15 minutes, 30 minutes, and 1 to 8 hours, when following the empirical transition probabilities. In effect, this is equivalent to the following: (1) consider a single taxi located at any initial position, (2) compute the probabilities that this taxi will be located at any given position after a set period of "mixing", and (3) compare this distribution to the overall taxi distribution at that time. Note that we consider the worst-case possible initial conditions (and thus our results apply to any initial taxi distribution or individual taxi location).

Let $\vec{\pi}_t = (\pi_1, \pi_2, \ldots, \pi_r)^T$ be the overall taxi distribution at time t, and $\vec{x} = (x_1, x_2, \ldots, x_r)^T$ be any initial distribution of taxis, where r is the number of regions used for analysis. Let $A^*_{u \to v}$ be the transition probabilities matrix from time u to v. Note that the transition probabilities matrix is estimated using taxi data and that we do not make any Markovian assumptions in its construction.

Hence, $\vec{\pi}_t^T = \vec{\pi}_0^T A^*_{0 \to t}$ is the overall distribution at time t. The time-t distribution obtained from an initial distribution of $\vec{x}$ is $\vec{x}^T A^*_{0 \to t}$.

We now present an upper bound on the L^∞ distance between any initial taxi distribution and the observed overall taxi distribution after a given time t.

Claim: $||\vec{x}^T A^*_{0\to t} - \vec{\pi}_t^T||_\infty \leq \max_{i,j} ||(\vec{e}_i - \vec{e}_j)A^*_{0\to t}||_\infty$, where $\vec{e}_i$ is the standard basis vector $(0,\ldots,0,1,0,\ldots,0)$, and $||\cdot||_\infty$ is the L_∞ norm, returning the maximum absolute element of a vector.

Proof: Let $[\vec{y}]_k$ denote the k element of any given vector $\vec{y}$. Note that $\vec{x} = \sum_i x_i \vec{e}_i$, so:

$$\begin{aligned}
\left[\vec{x}^T A^*_{0\to t}\right]_k &= \left[\sum_i x_i \vec{e}_i^T A^*_{0\to t}\right]_k = \sum_i x_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k \\
&\leq \sum_i x_i \max_j \left[\vec{e}_j^T A^*_{0\to t}\right]_k = \left(\sum_i x_i\right) \max_j \left[\vec{e}_j^T A^*_{0\to t}\right]_k \\
&= \max_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k
\end{aligned}$$

$$\begin{aligned}
\left[\vec{x}^T A^*_{0\to t}\right]_k &= \left[\sum_i x_i \vec{e}_i^T A^*_{0\to t}\right]_k = \sum_i x_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k \\
&\geq \sum_i x_i \min_j \left[\vec{e}_j^T A^*_{0\to t}\right]_k = \left(\sum_i x_i\right) \min_j \left[\vec{e}_j^T A^*_{0\to t}\right]_k \\
&= \min_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k
\end{aligned}$$

which bounds

$$\min_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k \leq \left[\vec{x}^T A^*_{0\to t}\right]_k \leq \max_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k$$

Similarly, we can show for $\vec{\pi}_t^T = \vec{\pi}_0^T A^*_{0\to t}$:

$$\min_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k \leq \left[\vec{\pi}_t^T\right]_k \leq \max_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k$$

Thus:

$$\begin{aligned}
&\min_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k - \max_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k \\
\leq\ & \left[\vec{x}^T A^*_{0\to t} - \vec{\pi}_t^T\right]_k \\
\leq\ & \max_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k - \min_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k
\end{aligned}$$

$$\begin{aligned} \Longrightarrow \quad & |\left[\vec{x}^T A^*_{0\to t} - \vec{\pi}_t^T\right]_k| \\ \leq \quad & |\max_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k - \min_i \left[\vec{e}_i^T A^*_{0\to t}\right]_k| \\ = \quad & \max_{i,j} |\left[(\vec{e}_i - \vec{e}_j)^T A^*_{0\to t}\right]_k| \end{aligned}$$

$$\Longrightarrow ||\vec{x}^T A^*_{0\to t} - \vec{\pi}_t^T||_\infty \leq \max_{i,j} ||(\vec{e}_i - \vec{e}_j)^T A^*_{0\to t}||_\infty \; \square$$

Note that the above upper bound $\max_{i,j} ||(\vec{e}_i - \vec{e}_j)^T A^*_{0\to t}||_\infty$ holds for any initial distributions $\vec{x}$ and $\vec{\pi}_0$. In other words, ***any*** two initial distributions will converge to within a L_∞ distance of $\max_{i,j} ||(\vec{e}_i - \vec{e}_j)^T A^*_{0\to t}||_\infty$ of each other within t time-steps. Hence, to show that the taxi distributions converge regardless of initial locations, we need to show that this upper bound decreases quickly as t increases.

This upper bound has a very natural interpretation. The distribution $\vec{e}_i^T A^*_{0\to t}$ is obtained when a subset of taxis concentrated in a single region is tracked for t time-steps. This initial configuration is arguably the worst-case scenario, which we intuitively expect to take the longest time to converge. The upper bound is thus a measure of the worst-case scenario, where two extreme initial distributions are allowed to converge over t time-steps.

Figure 7-5 shows the results of our analysis for workdays and non-workdays. Each curve represents a different time period such as "6 hours" (red curve with crosses); this corresponds to the amount of time any initial taxi distribution (or individual taxi) is allowed to "randomly walk" before its posterior distribution is compared to the overall taxi distribution. The y-axis gives the upper bound on the L^∞ distance between the "random-walked" taxi distribution and the empirical overall taxi distribution. For instance, at 09:30 on a workday, the upper bound on the L^∞ distance after two hours of "random-walks" is approximately 0.25, indicating that starting from 07:30, any two initial taxi distributions will come within an L^∞ distance of 0.25 of each other by 09:30.[2] Note that the convergence rate is considerably worse in the early morning hours, as there is far less taxi traffic in the middle of the night. As can be seen, we have fairly rapid convergence during almost all waking hours.

In our analysis, these upper bounds represent extreme, worst-case scenarios: the taxis

[2] These values are the *average*, over all work or non-work days, of the *worst-case*, over all initial taxi distributions, for the time period of interest.

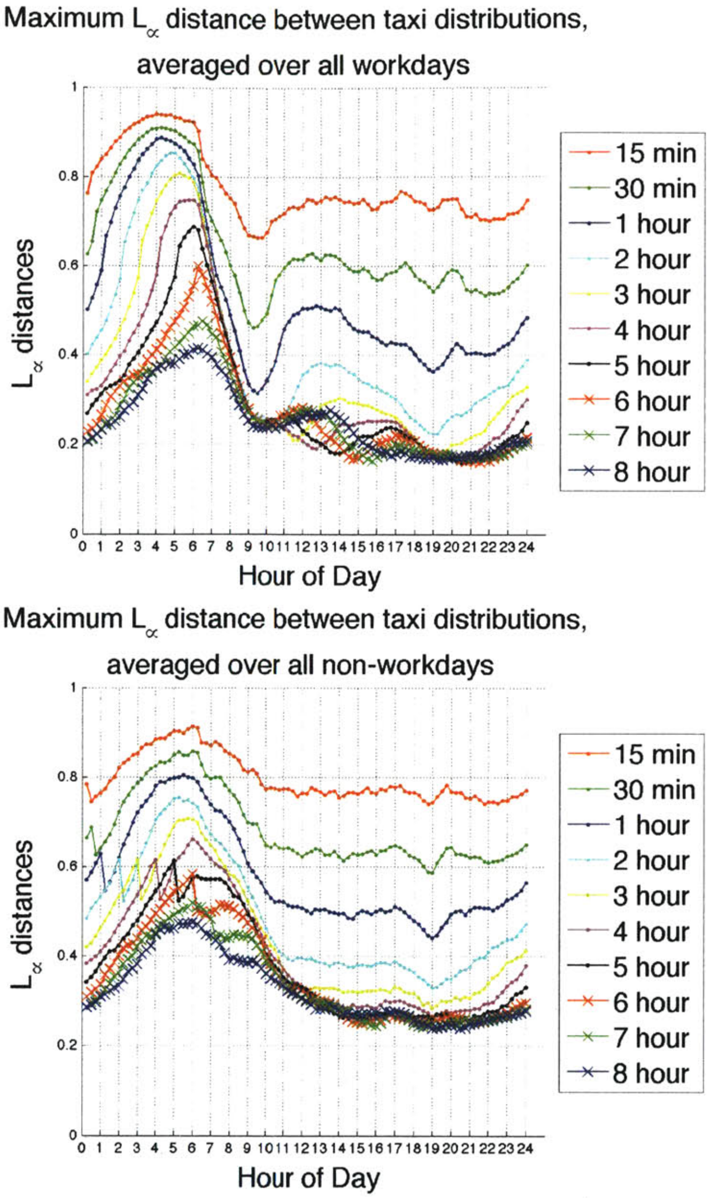


Figure 7-5: Upper bound on L^∞ distance between distributions of taxis with different initial distributions, after a given period of time. Within a short amount of time, low upper bounds on the L^∞ distances are attained, indicating that the taxi distributions have converged, so the taxi probes provide good coverage regardless of the initial distribution.

are concentrated in a single region, thus providing little coverage of the country. The low upper bound values demonstrate that any initial distribution of taxis will quickly disperse over the country to provide coverage on all regions.

7.4 Synopsis

In this chapter, we introduced our congestion-aware route planning system and the data sources we use for the system. Our analysis shows that the taxi data is a good source of data for collecting traffic information since taxis cover the entire area rapidly. In the following chapters, we show methods for processing these data to estimate necessary traffic information used for our traffic routing system.

Chapter 8

From Theory to Practice: Data-Driven Traffic Flow Estimation

In this chapter, we provide a method to accurately infer traffic flow through data collected from a roving network of taxi *probes* that log their locations and speeds at regular intervals. Traffic flow is defined as the number of vehicles that transit a road segment per unit time and is typically measured in units of vehicles per hour. Traffic flow is an important measure for traffic condition of a road.

The estimated traffic flow is used to detect congestion in Chapter 9. In addition, it is used for building the link-performance function for each road segment, which is used for evaluating the multi-agent routing system in Chapter 10.

This chapter is organized as follows. Section 8.1 describes our approach. Section 8.2 provides a method for inferring traffic flows from taxi probes. We train the inference model for locations where a small set of ground truth information exists and apply the model to predict the flow information for the same locations. Section 8.2.1 presents an analysis of how many devices are needed to achieve the inference within user-defined error limits. Section 8.3 develops an estimation method for traffic flows for locations where the ground truth traffic flow information does not exist.

8.1 Modeling and Predicting Traffic with Taxi Probes

Our hypothesis in this chapter is that a small number of dynamic probes are sufficient to characterize overall traffic at a city scale. Two natural questions arise: (1) How well does the data from dynamic probes represent overall traffic? (2a) If it is representative, how much data is needed to infer a good model for traffic? and (2b) If it is not representative, is the bias consistent and correctable?

Our approach to these questions is based on two types of sensor data: static and dynamic. Static sensors are placed at a fixed location to collect information about traffic as it passes by, for example, loop detectors or traffic cameras. Dynamic sensors are attached to the vehicles themselves and collect information about individual vehicles as they move. Note that with this setup, the dynamic data is strictly richer than the static data in the sense that the static data can be *inferred* from the dynamic data, but not vice versa. Why? Suppose that every vehicle was outfitted with a dynamic sensor and every intersection was outfitted with a static sensor. The static sensors are effectively collecting *macro-level data*, such as the number of vehicles passing through a given intersection during a given time interval, and this information can be inferred from the *micro-level data* collected by the dynamic vehicle sensors, for example, by simply counting the number of such vehicles whose sensed routes pass through the given intersection during the given time interval. However, the micro-level dynamic data cannot be inferred from the static sensor data, unless individual vehicles can be identified.

Now suppose that we did not have every vehicle outfitted with a dynamic sensor, but we did have a perfect random sample of such vehicles so outfitted. Then the static data inferred from the dynamic data would not be correct in *absolute* terms, but it would be correct in *relative* terms. In other words, if $1/3$ of the vehicles (chosen uniformly at random) were equipped with dynamic sensors, then the static traffic data inferred should be 1/3 of the actual data collected from the static sensors. The *distribution* of traceffic should be correct.

This observation provides a mechanism for quantitatively testing how representative the probe data is: Given any set of static sensor measurements, infer those measurements from the dynamic probe data and compare the results in relative terms, e.g., by comparing the corresponding traffic distributions and/or traffic flows. If the probe data is representative, these distributions and/or flows should match; if not, then one can quantify the mismatch, identify specific areas of match and mismatch, correct for consistent biases, and so on. Note that one would naturally suspect that probe data is not generally representative of all traffic: for example taxis do not ply the same routes as trucks. But we can qualify and quantify this mismatch.

There may be areas of Singapore where the taxi data is quite representative, for example, in the Orchard Road area. Here we can use the taxi data to quantify the *amount* of taxi data needed to infer good models of traffic: one can sub-sample the taxi data and see how the inferred models of traffic degrade vs. the gold-standard static sensor data.

8.2 Inferring Traffic Flow Using Taxi Probes Trained by Loop Detector Data

We employ machine learning, and in particular, a cross-validation study, to determine the simplest corrective model for inferring the loop distribution from the taxi distribution. We extend this result with a cross-validation study that shows that general traffic flow can also be inferred from the taxi data. Figure 8-2 shows the results of learning the corrective coefficients for taxi data and demonstrates that that taxi data can indeed be used to predict general traffic. Our analysis shows that the best model for accurately inferring traffic distribution and flow uses (1) the hour of the day and (2) whether the day is a workday or non-workday.

Let R be a set of roads in our study. We selected the 1000 road segments from the Singapore road network whose lanes are equipped with loop detectors: $R = \{r_1, r_2, \cdots, r_{1000}\}$. For road r and time slot t we normalize the traffic value relative to the overall traffic, to determine the fraction of traffic at that location at that time. Specifically, if t_r and l_r are the vectors representing the taxi and loop detector counts for each 15 minute time slot for road

r^1 and $\tilde{t}_r$ and $\tilde{l}_r$ are the corresponding distribution of taxis and loop detector counts, then

$$\tilde{t}_r(t) = \frac{t_r}{\sum_{i=1}^{1000} t_i(t)},\ \tilde{l}_r(t) = \frac{l_r}{\sum_{i=1}^{1000}} l_i(t)$$

Our goals are (1) to examine if the $\tilde{t}_r$ can be used to infer $\tilde{l}_r$ and (2) how well we can predict $\tilde{l}_r$ from $\tilde{t}_r$. To compile the relationship between $\tilde{l}_r$ and $\tilde{t}_r$, we used a logistic regression model as follows:

$$\tilde{l}_r = \frac{1}{1 + e^{-(\beta_0 + \beta_1 \tilde{t}_r)}},$$

where β_0 and β_1 are the two regression parameters we will learn.

We will find the best categorization of time slots for learning β_0 and β_1 that result in the best prediction of $\tilde{l}_r$, using the day-of-week and time-of-day categorization.

The intuition behind this is from the observation of the traffic distribution pattern. We saw significant periodic patterns based on day and week. Specifically, our day-of-week categories include ”Each Day of Week”, ”Workday/Non-workday”, and ”All Days together”, where non-workday means Saturday, Sunday or holiday. Our time-of-day category varies from 15 minutes to 24 hours. For example, Workday/Non-workday as day-of-week category and 2 hours as time-of-day category results in $2 \times 12 = 24$ pairs of regression parameters.

We divided the 4 weeks (August 01∼28, 2010) data into 3-week testing sets and a 1-week training set. We learn the regression parameters using the training set and apply the parameters to the left-out test set to find the test error. Figure 8-1 shows the leave-one-out cross-validation training and testing RMSE errors after doing the logistic regression for a road segment. As expected, the training error decreases when we use more complex models; it decreases as the number of time-of-day slots increase, and as the number of day-of-week slots increase. However, the testing error does not always decrease as the model complexity increases (generally known as over-fitting.) Figure 8-1 shows that the best test error was achieved at Workday/Non-workday for 15 minutes and the test error is not too large though we use a simpler model such as 1 hour. Figure 8-2(a) shows that using taxi data and the Workday/Non-workday 1 hour time model for learning the parameter we can

[1] Since we use 4 weeks of data, the length of t_r and l_r is 2688.

predict with highly accuracy general traffic distribution.

A similar analysis was done to infer traffic flow, represented as counts, from the dynamic probes using linear regression. The best test error was achieved for the model Workday/Non-workday and 1 hour time slot Figure 8-2(b) shows that using taxi data and the Workday/Non-Workday 1 hour time model for learning the parameters, we can predict with highly accuracy general traffic count.

The best test error was achieved at Workday/Non-workday for 15 minutes and the test error is not very large though we use a simpler model such as 1 hour. Figure 8-2(a) shows that using taxi data and the Workday/Non-workday 1 hour time model for learning the regression parameter we can predict with high accuracy the general traffic distribution.

A similar analysis was done to infer traffic flow, represented as counts, from the dynamic probes using linear regression. The best test error was achieved for the model Workday/Non-workday and 1 hour time slot. Figure 8-2(b) shows that using taxi data and the Workday/Non-Workday 1 hour time model for learning the parameters, we can predict with highly accuracy general traffic count.

Let us call RMSE as *absolute RMSE* and define the *relative RMSE* as the coefficient of variation of the RMSE as follows: relative RMSE $= \frac{\text{absolute RMSE}}{\text{mean of loop detector distribution}}$. To see how representative the taxi data is for the general traffic in Singapore, we computed the relative RMSE for all 1,155 road segments and observed that the relative RMSE is less than 10% for vast majority (80.3%) of the road segments.

We next show empirically that accurate estimates of the general traffic flows are possible using only a small number of probes.

8.2.1 How Many Taxi Sensors Are Needed?

Since taxi sensor data can be used to infer general traffic flow, we conclude that a taxi sensor network is a good proxy for general traffic and would like to know how many nodes are needed to infer traffic within some desired accuracy. There are several attributes that tradeoff (1) the number of nodes; (2) the amount of time the nodes roam to collect historical data; (3) the fraction of roads the nodes cover, and (4) the accuracy of the general traffic

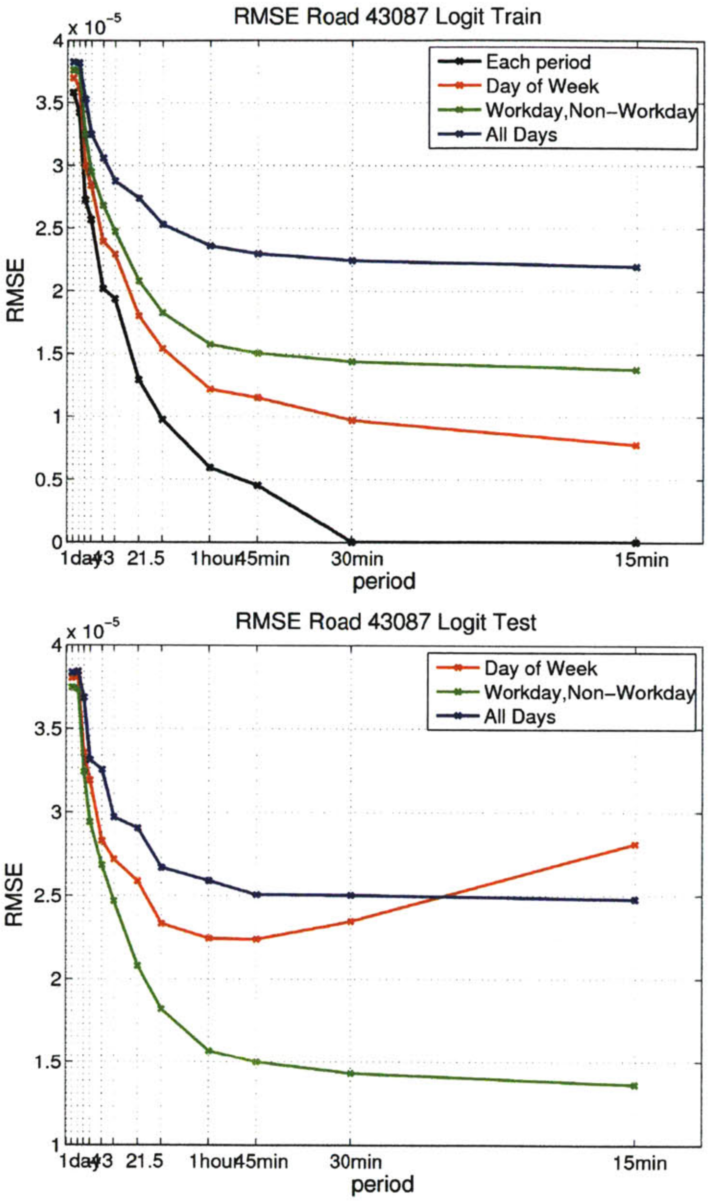
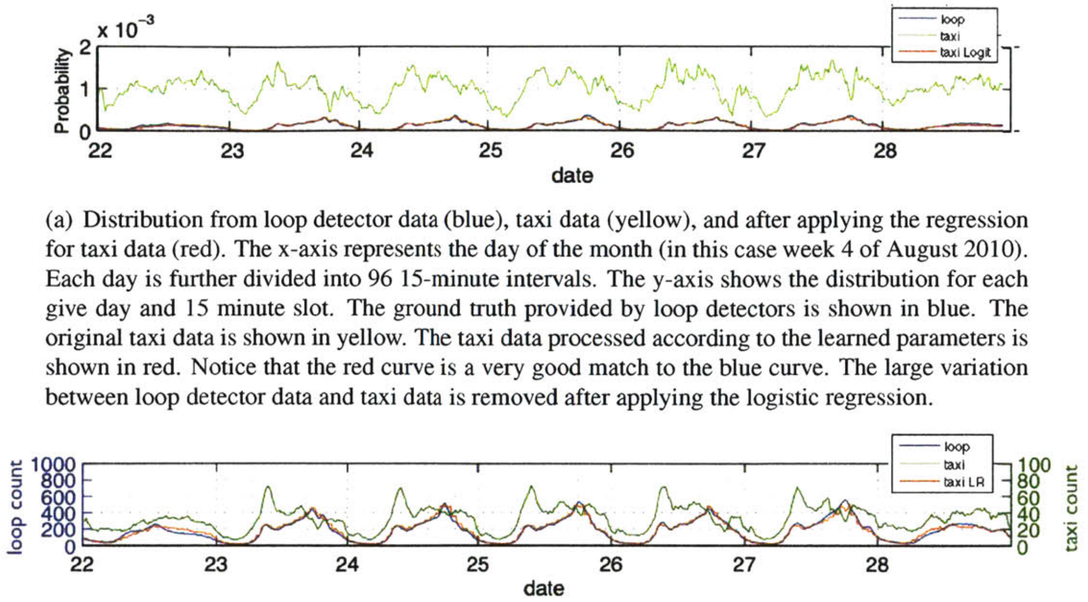


Figure 8-1: Training(left plot) and test(right plot) RMSE for a road. The x-axis represents the time window considered. The y-axis represents the RMSE. We can observe that the training error decreases as the model gets more complex, but the test error shows overfitting effect. The best category for this road is Workday/Non-workday and 15minutes.

prediction from the taxi sensor network nodes. In this section we show that a relatively small number of taxis is sufficient to get good road coverage, and that if the taxis roam for a relatively small amount of time, general traffic can be inferred with high accuracy as



(a) Distribution from loop detector data (blue), taxi data (yellow), and after applying the regression for taxi data (red). The x-axis represents the day of the month (in this case week 4 of August 2010). Each day is further divided into 96 15-minute intervals. The y-axis shows the distribution for each give day and 15 minute slot. The ground truth provided by loop detectors is shown in blue. The original taxi data is shown in yellow. The taxi data processed according to the learned parameters is shown in red. Notice that the red curve is a very good match to the blue curve. The large variation between loop detector data and taxi data is removed after applying the logistic regression.

(b) Count from loop detector data (blue), taxi data (green), and after applying the regression for taxi data (red). Whereas logistic regression was used for regression of distributions, linear regression was naturally used for count regression.

Figure 8-2: Result of regression

measured by RMSE.

Coverage vs. Number of Taxis

Section 7.3 shows that taxis travel randomly and therefore they cover Singapore broadly. In this section we quantify this property of taxi movement by investigating the fraction of road segments they cover. We measure *coverage* as the fraction of roads that were driven on at least k times[2] by any taxi. We examined the coverage for different time windows and different k, using sets of taxis of size N that were randomly selected using the following procedure. We generate a random permutation of 16,000 taxis and use the first N taxis from the permutation sequence.

Figure 8-3 was drawn for a single permutation of 16,000 taxis. The plots show the fraction of the 1,000 road segments driven on least 30 times during 2 different time windows. The number of taxis used for the coverage is given by x-axis. Different curves correspond

[2] k is a parameter we choose to support the methodology of inferring general traffic from the taxi counts.

to different times of day, as denoted in the legend. The time window is 15 minutes (left) and 1 hour (right). For workday coverage during 15 minutes (left plot), we conclude that 700 taxis are enough to cover 70% of the roads for most of the day's 1-hour time windows except those in the middle of the night when the number of vehicles on the road is sparse. Figure 8-4 shows cumulative histograms of 300 random permutations. The steep slopes indicate that the coverage result is not very different from one permutation sequence to another. From the right plot, we can see that 2000 taxis are enough to cover 90 % of the total loop detector locations during 08:00 ~ 08:15 on all the workdays.

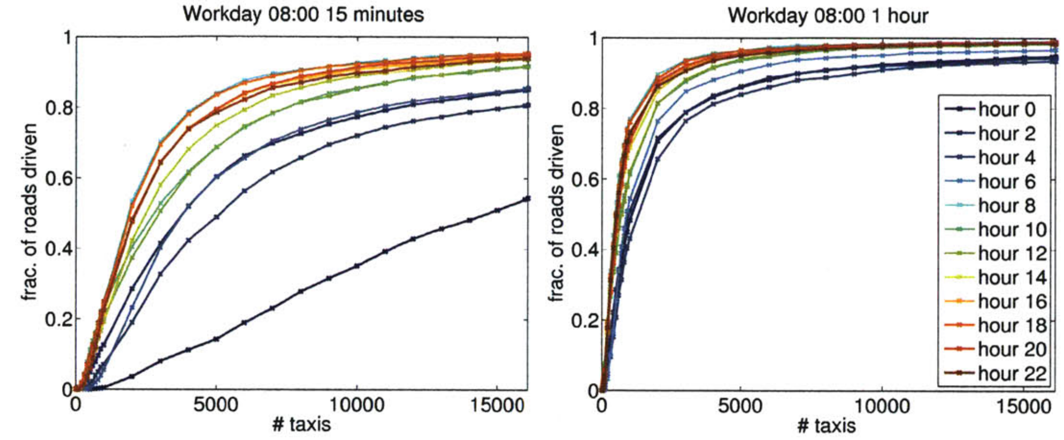


Figure 8-3: Coverage according to the number of probes. The Left and right plots show what fraction of roads are covered 30 times during 15 minutes and 1 hour respectively (y-axis) as the number of probes grows (x-axis). Color encodes the time of day for the coverage measurement.

RMSE vs. Number of Taxis

Next, we consider how many taxis are needed in order to estimate traffic on the Singapore road network within a desired RMSE. We use as base case for traffic estimation the RMSE derived in Figure 8-2(a) using the entire fleet of 16,000 taxi probes and ask, how much will the RMSE degrade if we use a subset of size given by parameter k of the the probes selected randomly. Figure 8-5(a) shows typical data for the RMSE. The RMSE decreases as the number of taxi probes increases. However, the RMSE bottoms out at a certain point so that increasing number of probes has no effect on improving the RMSE value. Figure 8-5(a)

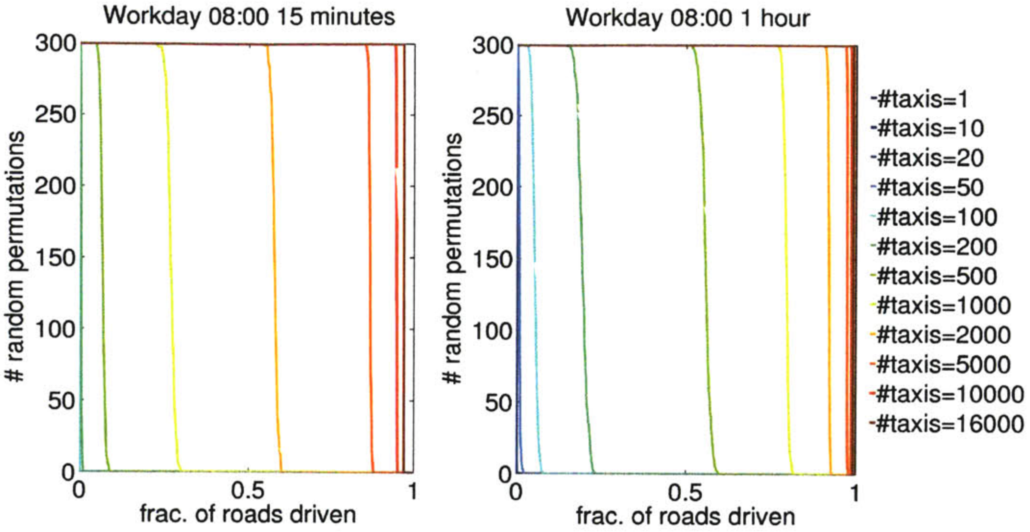
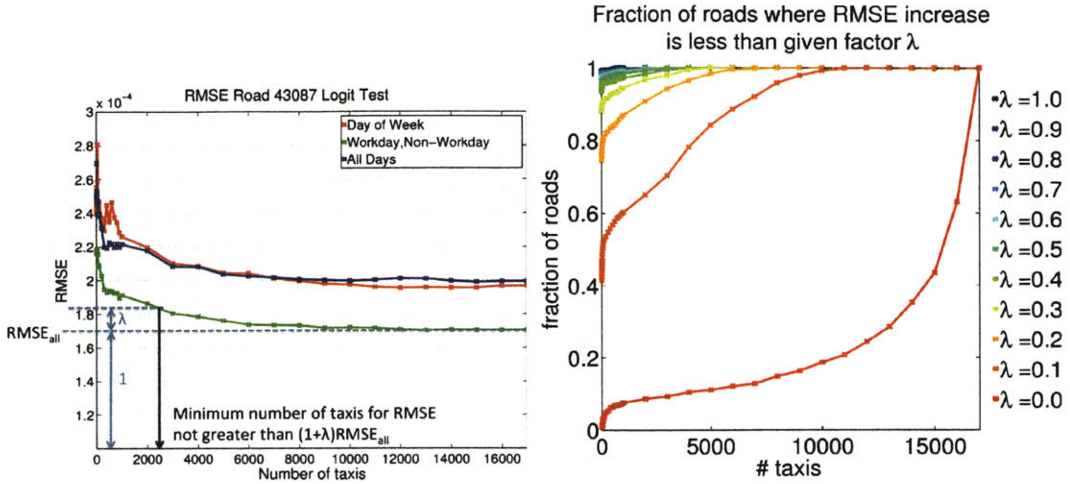


Figure 8-4: Cumulative histogram of coverage by 30 samples for 300 random permutations for 8am. The steep slope of each line means that the variation from one random order selection of 16000 taxis are not very different from the others. Thus, we don't need to worry too much about which set of taxis we should choose

shows the tradeoff between the number of taxis used for the prediction and the accuracy of the prediction.

Let $RMSE_T$ denote the RMSE achievable with our entire fleet of 16,000 dynamic probes and let $n_\lambda(c)$ be defined as the minimum number of taxis that can be used to infer general traffic with RMSE at most $(1+\lambda)RMSE_T$ for a given traffic model c. Figure 8-5(a) shows $n_\lambda(c)$. Figure 8-5(b) shows a cumulative histogram of $n_\lambda(c)$ for various λ values where c is the model workday/non-workday with a 1 hour time slot. The y-axis of both plots represents the fraction of road segments among the total 1,000 road segments. The left plot is drawn over *# probes* for various λ, and the right one is drawn over λ for various *# taxis*. In Figure 8-5(b) the increase in percentage is steep from 0 to 1000, which shows that increasing the number of probes from 0 to 1,000 improves estimation. Given 2,000 probes we can estimate the general traffic counts as given by the loop detectors for 65% of the road network with precision at least $1.1RMSE_T$.



(a) RMSE of a road over different number of taxi probes. The fewer probes we use, the higher RMSE gets. This graph shows a tradeoff between accuracy and number of probes. λ defines the tolerance for accuracy in RMSE for using fewer probes.

(b) Cumulative histogram of $n_\lambda(c)$ Each curve in the plot corresponds to a different λ. The x-axis plots the error tolerance, and the y-axis plots the percentage of roads With 2000 probes and 10% additional error over $RMSE_T$ (the orange curve), we can predict the traffic flows for 65% of the roads

Figure 8-5: The dependence of the RMSE for inferred traffic flows on the number of dynamic probes used.

8.3 Estimating Traffic Flow using Taxi Probes and Inferred Traffic Flow for Limited Locations

Whereas travel time data for each road segment can be observed by taxis, the flow data is only available for the locations where loop detectors exist. In this section we describe how to estimate flow for each road segment. We develop a computational approach to estimating the flow for locations where the loop data is not available. We estimate the traffic flow using the predicted loop count learned in Section 8.2.

Let O be the set of all roads with associated loop detectors. Suppose road i does not have an associated loop detector ($i \notin O$). Our goal is to estimate the flow for road i using the real-time flow data from all the roads with associated loop detectors O and taxi count information for the road set $O \cup \{i\}$. Intuitively, we will estimate the flow of road i by the weighted average of inferred flow by a method given in Section 8.2 at other roads with loop detectors, where the weight is defined by the similarity between road i and a road j in O.

```
Algorithm 15: Estimate-Flow
Data: $f(j)$: inferred loop count for road $j$ where loop detector exists;
      $f_t(i)$: taxi count data for road $i$;
      euclidean distance between all pairs of roads;
      angular difference of orientation between all pairs of roads;
      difference in number of lanes between all pairs of roads;
      difference in taxi count for all pairs of roads;
      $u_k$: the indicators for each similarity measure;
Result: $\tilde{f}(i)$: estimated relative flow for road $i$ without loop detector
$O$ = a set of all roads where loop detectors exist ;
for $j \in O$ do
    Find $m_1(i,j)$, $m_2(i,j)$, $m_3(i,j)$, and $m_4(i,j)$ ;
    $s(i,j) = m_1(i,j)^{u_1} \times m_2(i,j)^{u_2} \times m_3(i,j)^{u_3} \times m_4(i,j)^{u_4}$;
    $w_i(j) = \frac{s(i,j)}{\sum_{\forall k \in O} s(i,k)}, \forall j \in O$
end
$\tilde{f}(i) = (f_t(i)) \sum_{\forall j \in O} w_i(j) \frac{f(j)}{f_t(j)}$;
```

We quantify the similarity between roads i and j by several measures:

- measure 1: $m_1(i,j) = \frac{1}{d(i,j)}$, where $d(i,j)$ is the Euclidean distance between road i and road j
- measure 2: $m_2(i,j) = \frac{1}{a(i,j)}$, where $a(i,j)$ is the angular difference between the orientation of road i and that of road j
- measure 3: $m_3(i,j) = \frac{1}{l(i,j)}$, where $l(i,j)$ is the difference between number of lanes of road i and that of road j
- measure 4: $m_4(i,j) = \frac{1}{t(i,j)}$, where $t(i,j)$ is the difference between taxi count for road i and that of road j

Each of these four measures is an inversely proportional relation. Measures 1, 2, and 3 are static. The information depends on neither time nor traffic conditions. Measure 4 captures the dynamic real-time information observed by taxi probes.

We define an aggregated similarity measure using the four measures. For a pair of roads

i and j, (i,j), the aggregate similarity measure is given as follows:

$$s(i,j) = m_1(i,j)^{u_1} \times m_2(i,j)^{u_2} \times m_3(i,j)^{u_3} \times m_4(i,j)^{u_4} \tag{8.1}$$

where u_k is the indicator for whether measure m_k should be considered for deciding the aggregate similarity.

$$u_k = \begin{cases} 1 & \text{if measure } m_k \text{ should be considered} \\ 0 & \text{otherwise} \end{cases}$$

We choose the best u_k empirically.

Algorithm 18 describes the method for real-time estimation of traffic flow $\tilde{f}(i)$ for road $i \notin O$, using the real-time loop count estimation from all roads in O. Our algorithm estimates the flow by the weighted average of the available inferred loop detector counts for all the roads in O. The weight assigned to each loop detector on road j for the estimation of flow for road i, $w_i(j)$, is defined as the normalized aggregate similarity measure for all the road segments in O as follows:

$$w_i(j) = \frac{s(i,j)}{\sum_{\forall k \in O} s(i,k)}, \exists j \in O \tag{8.2}$$

The estimation of flow for i, $\tilde{f}(i)$, is calculated as the weighted average of the other inferred loop counts using the weight $w_i(j), \forall j \in O$ as follows:

$$\tilde{f}(i) = (f_t(i)) \sum_{\forall j \in O, f_t(j) \neq 0} w_i(j) \frac{f(j)}{f_t(j)} \tag{8.3}$$

where $f(j)$ is the observed loop count of road j divided by the number of lanes of road j.

8.3.1 Quality of Estimation

In order to evaluate performance given our data (Section 7.2) we conduct a leave-one-out cross-validation approach. We selected 1,000 locations with loop detector data as a test set, O. We use each road segment $i \in O$ as a test case. For each i we define $O_i = O \setminus i$ to contain all the road segments with loops. We test our algorithms for congestion detection for i using O_i. We use the loop counts on i as ground truth to quantify our estimates. The data consists of loop detector data for August 2010 and taxi data for the same period, where the loop counts and taxi speeds are collected for each 15-minute time slot. Thus, we have the loop counts and taxi speeds data for 1,000 locations with 31 days × 96 slots per day = $2,976$ data points. Using these data, we ran our Algorithms 18 for 1,000 test cases.

For each road, we estimated the flow using Algorithm 18. We used all possible combinations of similarity measures defined in Section 8.3 (a total of 16 combinations). The best estimation was obtained when we used all the measures with $(u_1, u_2, u_3, u_4) = (1, 1, 1, 1)$. Figures 8-6 and 8-7 show the quality of flow estimation. The estimated flow and the real flow have a high linear correlation, and the gradient of the linear relationship is highly concentrated around one.

8.4 Synopsis

In this chapter we have shown that taxi probes can be used to infer traffic patterns. Because the movement patterns of taxis is similar to a rapidly mixing Markov chain, a fleet of taxis can cover a city-scale road network very fast. Using data from a study conducted in Singapore, we show that a relatively small number of taxis (700) is needed to cover about 70% of the road network in order to provide sufficiently accurate traffic models. Traffic, as measured by taxis, provides a biased representation of general traffic in city-scale road networks. However, our study shows this bias is consistent and we can learn the corrective parameters. Thus, we conclude that taxi probes can be used to provide accurate traffic forecasting using historical data from their past drives, and real-time traffic snapshots from their current drives.

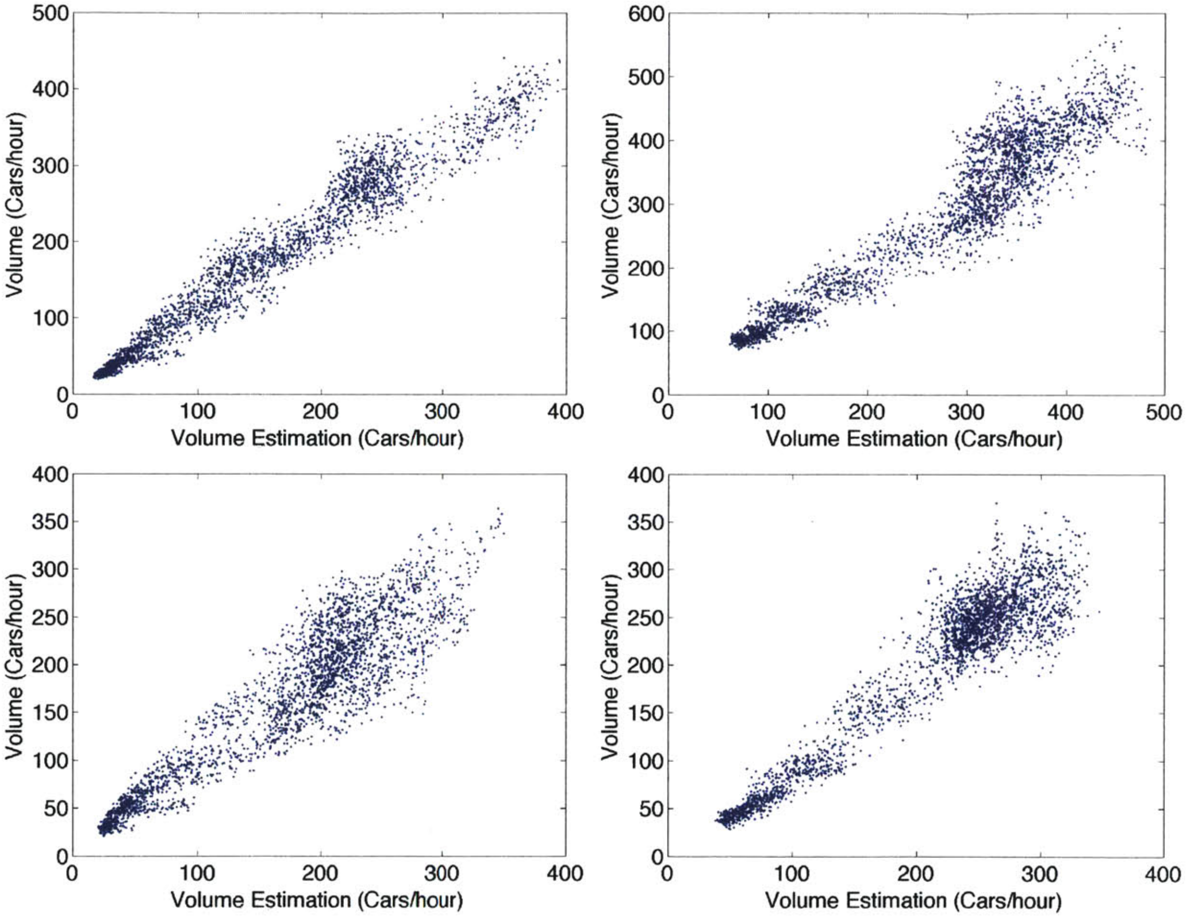


Figure 8-6: Estimated flow for four different roads. The estimated flow (x axis) is strongly linear to the real flow (y axis), and the gradient is very close to 1.

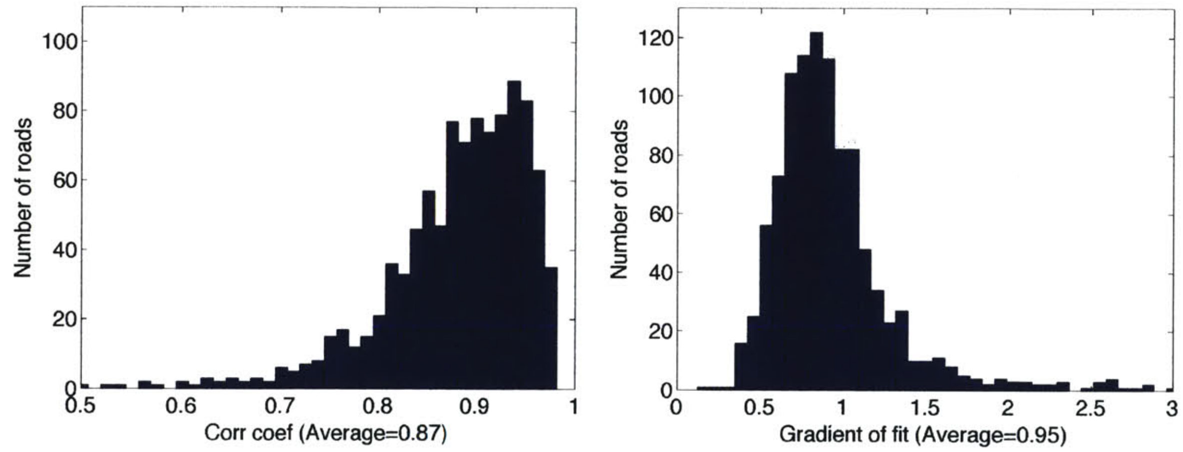


Figure 8-7: Quality of Flow Estimation. (left) Correlation coefficient of estimated flow and real flow. The correlation coefficient is high around 0.9 (right). 'Gradient' is the gradient of minimum-squared-error fit between real loop counts and the estimated loop counts. Most of the roads lie in the range of $0.7 \sim 1.3$. Thus, the gradient error is mostly in the 30% range. Thus, the estimated flow approximates well the real flow.

Chapter 9

From Theory to Practice: Data-Driven Congestion Detection

Traffic congestion is presently and primarily estimated via static sensors, such as traffic cameras and loop detectors. This information is often difficult to collect, aggregate, analyze and quantify, especially in real time. In this chapter, we demonstrate that it is possible to accurately infer traffic congestion through data collected from a roving network of taxi *probes* that log their locations and speeds at regular intervals. Our model and inference procedures can be used to *estimate* the natural capacity of a road or road segment off-line, as well as to *detect* whether a road has exceeded its capacity (and thus reached congestion) in real time. As such, our techniques provide a powerful new data source for traffic congestion analysis, dynamic road pricing, and urban planning.

This chapter is organized as follows. Section 9.1 provides a background on fundamental diagram. Section 9.2 describe our methods for estimating the critical point in the fundamental diagrams for each road. Section 9.3 presents the results of congestion detection. Section 9.4 discusses the discovery of operational states in addition to free flow and congestion.

9.1 Background: Fundamental Diagram

More specifically, the fundamental diagram of a road captures the relationship between flow, density, and speed on that road [106, 29, 35, 46, 41, 74, 15, 76, 136, 139, 140]. Flow is defined as the number of vehicles that transit a road segment per unit time and is typically measured in units of vehicles per hour; density is defined as the number of vehicles present on a road segment per unit distance and is typically measured in units of vehicle per kilometer; speed is defined in the usual way for vehicles on a road segment and is typically measured in units of kilometers per hour. The relationship between these three quantities is

$$\mathit{flow} = \mathit{speed} \times \mathit{density} \tag{9.1}$$

where maximum flow ultimately defines the capacity of a road segment.

The *fundamental diagram* for a road segment shows the tradeoff between speed vs. density, flow vs. density, and speed vs. flow (see Figure 9-1). As a general rule (1) speed decreases as density increases and (2) flow increases as density increases, until the degradation in speed caused by increasing density causes the flow ultimately to begin to decline. (Speed and flow have a non-functional relationship dictated by the above two rules and Eq (9.1).) The critical point in a flow-density curve is the point at which the maximum flow is reached: beyond this critical point, flow and speed will both decline as density increases, and a road segment will experience congestion.

The fundamental relation between speed, density, and flow for a given road segment will depend on different properties of the road (width of the lanes, grade), flow composition, external conditions (weather and ambient conditions), traffic regulations, etc. [106]. The fundamental diagram plays an important role in characterizing congestion [90, 98, 1, 92, 29, 81]. Prior research in understanding the fundamental diagram includes [92], where assessment methods for roadway capacity estimation are presented, and [29], where the estimation of the fundamental diagram is achieved through passing rate measurements in congestion.

Figure 9-1 shows the shape of the theoretical fundamental diagram and sample diagrams estimated by our study using flow measured by inductor loops and speed measured by processing taxi trajectory data. The density was calculated from these two measured quantities using Eq (9.1). The operating point of a road in the fundamental diagram is related to congestion. One challenge in computing the fundamental diagram and corresponding congestion point for any road segment is the availability of data. While a large fleet of taxis such as the Singapore fleet has comprehensive coverage of the road network and thus provides good speed data, flow information is only available for selected locations that have loop detectors.

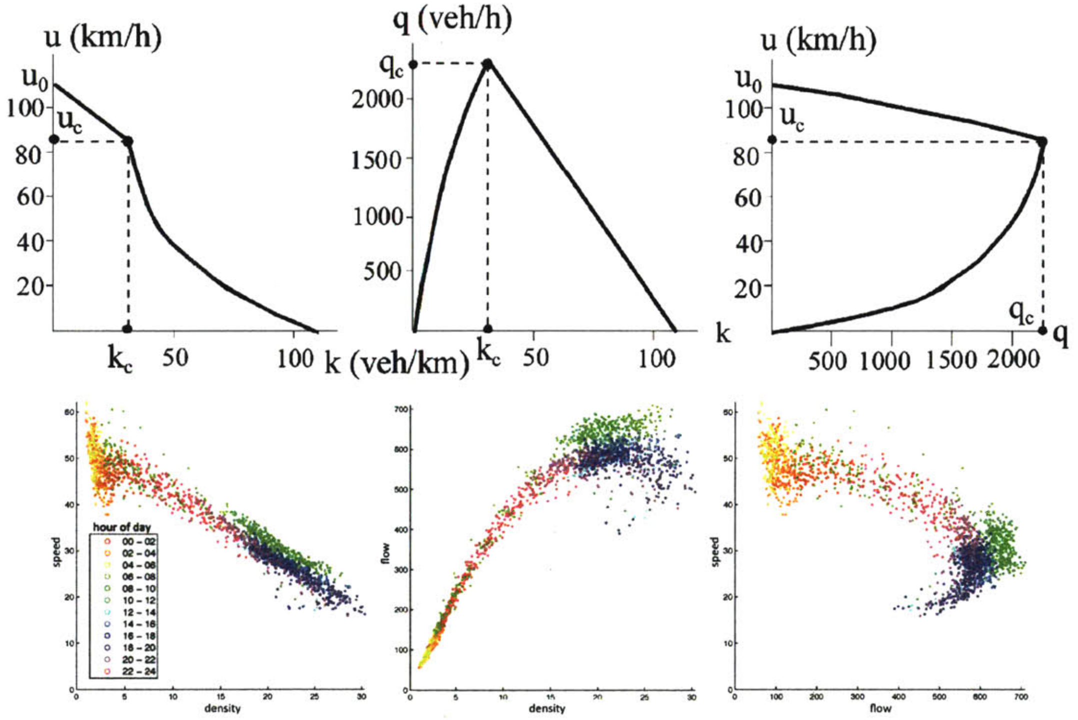


Figure 9-1: Examples of the fundamental relations between flow, density, and speed taken from [106] (top) and the empirical fundamental diagram constructed from the loop detector counts and taxi speed data for a sample road (bottom). Color indicates the time of day when the points were collected, as shown in the legend.

9.2 Congestion Detection

We assume that the road's real-time speed information is known. This assumption can be met using real-time taxi data, since the large fleet of taxis collects speed information for many road segments in parallel. We consider the speed information from the taxi probe vehicles as the ground-truth speed. Thus, we have the ground-truth speed information for every road.

We also assume that the real-time flow information is available for a subset of road segments. It is common for a city to have flow information from loop detectors and cameras for some of the road segments. In our evaluation, we use data from 1,000 loop detectors. We consider the loop detector count per unit time as the ground-truth flow information for the corresponding road[1].

Our goal in this section is to estimate the congestion state of a given road. This road may or may not have an associated loop detector. In Section 9.4 we generalize this goal to multiple operational states, including the heavy traffic state. Our approach has two steps. First, we build the estimation of the fundamental diagram from the available sensor data. Second, we estimate where in the estimated fundamental diagram the current road state lies, by matching the current road state to the estimated fundamental diagram.

We detail our computational solution for the two steps in the following subsections.

9.2.1 Estimation of the Critical Point in the Fundamental Diagram

Figure 9-2 shows the estimated fundamental diagram for two roads. We see that the shape of the estimated fundamental diagram matches the shape of the theoretical fundamental diagram. We also clearly see the presence of a critical point. We observe that it is challenging to detect the slope change after the critical point (thus it is challenging to immediately see the severe congestion in this data). Rather than detecting this event, we focus on detecting the 'start of congestion' or 'near congestion', which happens around the critical point. The points suggesting near-congestion are of great interest to traffic planners. Algorithm 16 describes the method for estimating the critical point as the break point for the road. We use the flow-density curve for detecting the break points. We check whether the experimental estimated fundamental diagram has a break point, and if there exists one, where it is located. If the break point does not exist, the road does not experience significant congestion.

[1] Thus, for a road with a loop detector, we are provided with the data points for the real fundamental diagram.

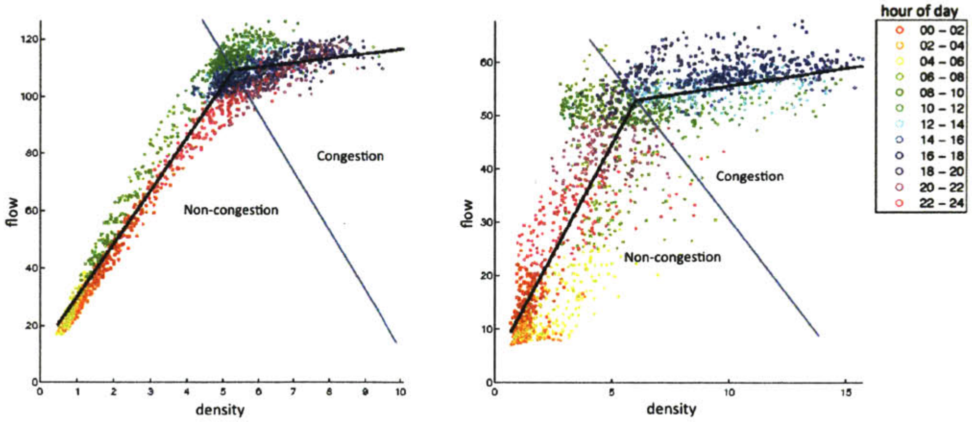


Figure 9-2: Example fundamental diagram of two different roads. Two-segmented piecewise linear function fitting the data is illustrated. The congestion and non-congestion regions are defined as the area divided by the line crossing the break point of the fitting curve. Color indicates the time of day when the points were collected, as shown in the legend.

The details of the algorithm are described in Algorithm 16. We fit the data points of the experimental flow-density fundamental diagram into piecewise linear segments using the method in [25, 108, 54] for fitting a curve to piecewise linear functions. We use a similar technique to find the best-fitting piecewise linear segments.

9.2.2 Classification of Congestion State

We provide a method for classifying the current state of a road as congestion or non-congestion. Once we find the break point, we classify each data point into congestion or non-congestion using the point's distance to the two line segments representing congestion or non-congestion, respectively. The classification of congestion state is illustrated in Figure 9-2. The dividing line is defined as the bisector of the angle between the two fitted linear segments. The data points lying in each region are classified according to the corresponding congestion state.

Algorithm 16: Find-Critical-Point

```
Data: $f$: a time sequence of estimated flow                $k$: a time sequence of
      estimated density
Result: break point $(k_{critical}, f_{critical})$
$k_{critical} = None$;
$f_{critical} = None$;
$cost_{min} = \infty$;
$f_{min}$ = minimum of $f$, $f_{max}$ = maximum of $f$;
$k_{min}$ = minimum of $k$, $k_{max}$ = maximum of $k$;
for $r \in \{0.05, 0.1, \cdots, 0.95\}$ do
    $x$ coordinate of break point $k_{bp} = r \times (k_{max} - k_{min})$ ;
    $(cost, slope, num, f_{bp})$ = fitLinearSegments($k$, $f$, $k_{min}$, $k_{bp}$, $k_{max}$) ;
    if $cost < cost_{min}$ and $slope < 0.2 \times \frac{f_{max} - f_{min}}{k_{max} - k_{min}}$ and $num > 200$ then
        $cost_{min} = cost$ ;
        $k_{critical} = k_{bp}$ ;
        $f_{critical} = f_{bp}$ ;
    end
end
```

Algorithm 17: Fit-Linear-Segments

```
Data: density $k$, density $f$, minimum of density $k_{min}$, $k_{bp}$, $k_{max}$
Result: $cost$ , $slope$, $num$, $f_{bp}$
Find the fitting two linear segments that minimizes the residual sum of squared
errors using the algorithms in [25, 108, 54] ;
$cost$ := residual sum of squared errors ;
$slope$ := the slope of the second linear segment ;
$num$ := the number of data points corresponding to the second linear segment ;
$f_{bp}$ := the $y$ coordinate of the break point found for $x = k_{bp}$
```

9.3 Experiments

In this section we describe our experiments with relative flow computation and critical point detection for roads that do not have associated loop detectors. In order to evaluate performance given our data (Section 7.2) we conduct a leave-one-out cross-validation approach. We selected 1,000 locations with loop detector data as a test set, O. We use each road segment $i \in O$ as a test case. For each i we define $O_i = O \setminus i$ to contain all the road segments with loops. We test our algorithms for congestion detection for i using O_i. We use the loop counts on i as ground truth to quantify our estimates. The data consists of loop detector data for August 2010 and taxi data for the same period, where the loop counts and taxi speeds are collected for each 15-minute time slot. Thus, we have the loop counts and taxi speeds data for 1,000 locations with 31 days × 96 slots per day = 2,976 data points. Using these data, we ran our Algorithms 18 and 16 for 1,000 test cases.

9.3.1 Quality of Flow Estimation

The quality of estimation of relative flow is shown in Figure 9-3 for four different estimation methods:

1. using taxi count as relative flow,
2. using loop count for estimation of relative flow with similarity measure 1 in Eq (8.1) ($(u_1, u_2, u_3, u_4) = (1,0,0,0)$),
3. using loop count for estimation of relative flow with similarity measures 1 through 3 in Eq 8.1 ($(u_1, u_2, u_3, u_4) = (1,1,1,0)$), and
4. using loop count for estimation of relative flow with similarity measures 1 through 4 in Eq (8.1) ($(u_1, u_2, u_3, u_4) = (1,1,1,1)$).

For each road we also used the taxi count as the estimation of the relative flow as a way of comparing our method with an alternative that does not use loop counts. Figure 9-3 shows the naive estimation using the taxi data (first column) followed by the relative flow estimation for $(u_1, u_2, u_3, u_4) = (1,0,0,0)$, $(1,1,1,0)$, and $(1,1,1,1)$.

9.3.2 Congestion Detection

Figure 9-3 also shows the relationship between the loop count data and the estimated relative flow. Figure 9-3 (1, 1) shows that the taxi count is not correlated to loop detector count very well. Thus, the fundamental diagram built using taxi speeds and taxi counts (Figure 9-3 top row) is quite different from the real fundamental diagram (Figure 9-3 bottom row).

Using the loop counts from other loop detectors, we can estimate the loop counts for the current road better. Figures 9-3 (2, 1), (3, 1), and (4, 1) show that the loop count estimation gets better when we use static information such as distance, orientation, and number of lanes, and the observed taxi count. When we use all the available static and dynamic information, the real loop counts and the estimated relative flow have a highly linear correlation. The histogram for the correlation coefficients for 1,000 roads is shown in Figure 9-4.

We can predict the relative flow for roads that do not have any loop detectors. The estimated fundamental diagrams in the fourth row of Figure 9-3 are very similar to the real fundamental diagrams in row 5. We conclude that the estimation for relative flow is accurate enough to identify the current operating point from the fundamental diagram.

Column 3 of Figure 9-3 shows the fundamental diagram used for congestion detection. We observe that the break points of the fitting linear functions for our estimated fundamental diagram are very similar to those for the ground truth (see the third column of Figure 9-3).

9.3.3 Congestion Detection Quality

Ground-truth congestion is determined as described in Section 9.2.2 using real loop detector counts instead of the estimated relative flows. We compare the ground-truth congestion and the estimated congestion, and show the detection probability (the probability that the real congestion is detected as congestion) and false-alarm probability (the probability that non-congestion is falsely detected as congestion) in Figure 9-5(b).

Figure 9-5(a) shows the detection and false-alarm probabilities for the four different estimation methods. When we use taxi counts as the relative flow, the false alarm rate and detection rate are approximately 0.5, meaning that the congestion detection using taxis is not much better than a random guess. The results show that we can predict congestion well when we use the real-time loop detector counts from other roads.

9.4 Detecting Heavy Traffic

In our modeling of the empirical fundamental diagrams, we have chosen to follow closely the theoretical relation between flow and density with a single critical point. However,

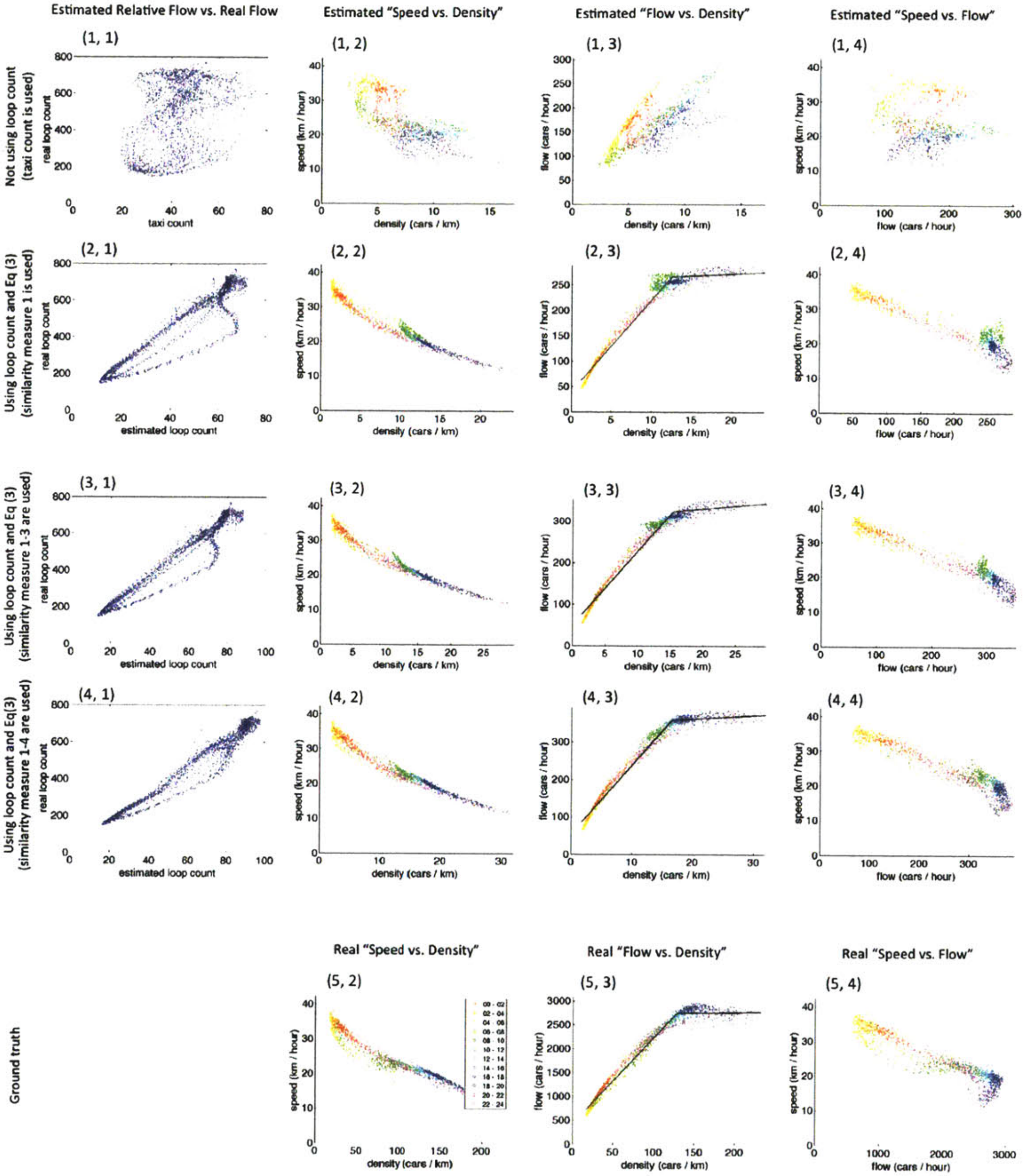


Figure 9-3: Each row shows the quality of estimation of relative flow and the estimated fundamental diagrams for each estimation method. For comparison, the bottom row shows the real fundamental diagrams built from the real loop count observations. [First column:] Real loop count (y-axis) vs. estimated relative flow (x-axis). [Second, third, and fourth columns:] speed-density, flow-density, speed-flow fundamental diagrams estimated from each method. The third column (flow-density relationship) is the one used for our congestion detection. Color indicates the time of day when the points were collected, as shown in the legend.

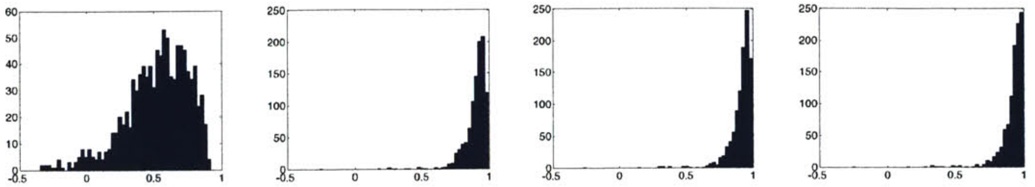
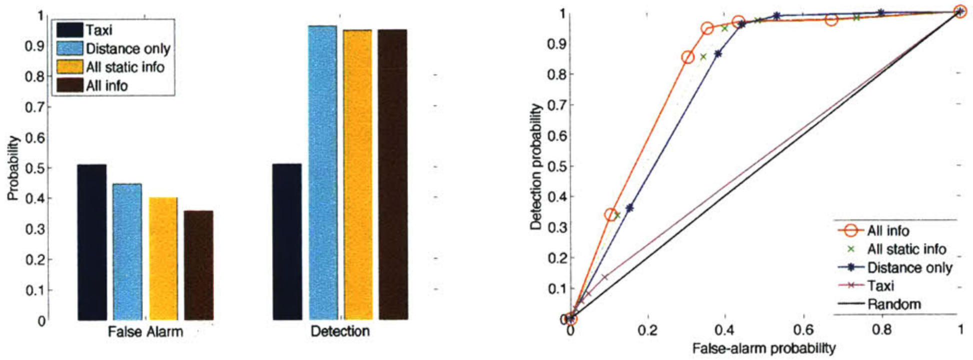


Figure 9-4: Histogram of correlation coefficient between loop count and loop count estimation. (x-axis: correlation coefficient; y-axis: number of roads). (1) taxi count is used, (2) similarity measure 1 is used, (3) similarity measure 1∼3 are used, (4) similarity measure 1∼4 are used.



(a) False alarm probability and detection probability. 'All info' has small false alarm probability and high detection probability.

(b) ROC curve. For each method, we ran multiple hypothesis tests by moving the congestion region division line, and got the ROC curve.

Figure 9-5: Quality of congestion detection

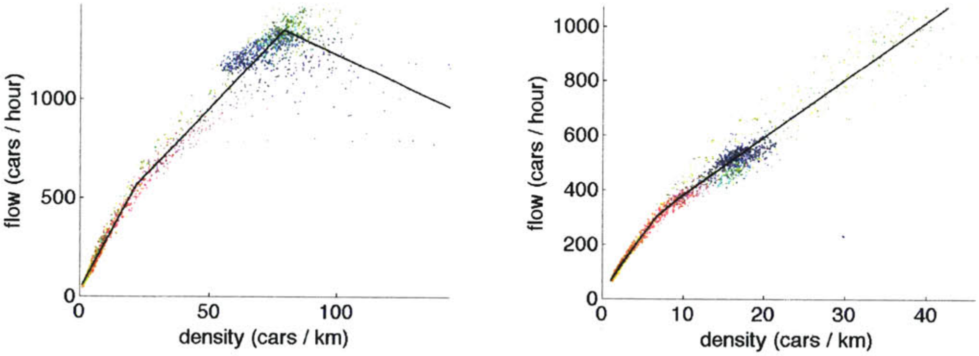
we observe that in the empirical fundamental diagrams, the line segment corresponding to the 'congestion' state often has a positive gradient, indicating that a higher throughput or flow may be obtained by increasing the car density on the road. This is contrary to the theoretical fundamental diagrams which postulate a negative gradient line past the critical point of congestion.

Therefore, we believe that such roads have not experienced 'true congestion' (where increased car density leads to lower throughput), but only 'heavy traffic'. The state of 'heavy traffic' is distinct from 'free flow', where flow increases rapidly with small increases in car density. The theoretical fundamental diagrams consist of only two operational states, and fail to account for additional states such as 'heavy traffic' that occur betwen 'free flow' and 'congestion'.

In order to discover these additional operational states, we ran the piecewise line-fitting algorithm on the data, with the number of line segments varied from 1 to 5. The model of best fit was then selected using a minimum description length (MDL) criterion, which provides a balance between optimizing the error and the complexity of the model. We adapt the polynomial regression MDL in [79] to our piecewise line fitting:

$$MDL = \frac{m+1}{N}\log_2(N) + \log_2(cost/N)$$

where N is the number of data points, $m \in \{1,2,3,4,5\}$ is the number of line segments, and *cost* is the residual sum of squared errors.



(a) Example of a road with three operational states of 'free flow' (low density), 'heavy traffic' (medium density) and 'congestion' (high density).

(b) Example of a road with two operational states of 'free flow' (low density) and 'heavy traffic' (medium density).

Figure 9-6: Piecewise line fitting to empirical fundamental diagrams, using the MDL criterion to select the number of line segments. Color indicates the time of day when the points were collected, as shown in the legend.

Figure 9-6(a) shows an example of the piecewise line fitting using the MDL criterion. The curve of best fit found consists of three line segments. As the car density is increased, the road progresses from 'free flow' (fast increase in flow as car density increases) to 'heavy traffic' (slower increase in flow as car density increases) and finally breaks down into 'congestion' (decrease in flow as car density increases).

On the other hand, Figure 9-6(b) shows a different road with only two operational states of 'free flow' and 'heavy traffic'. The gradients of both line segments are positive, indicating that the road has never experienced 'congestion' where it is no longer possible to increase flow by increasing density.

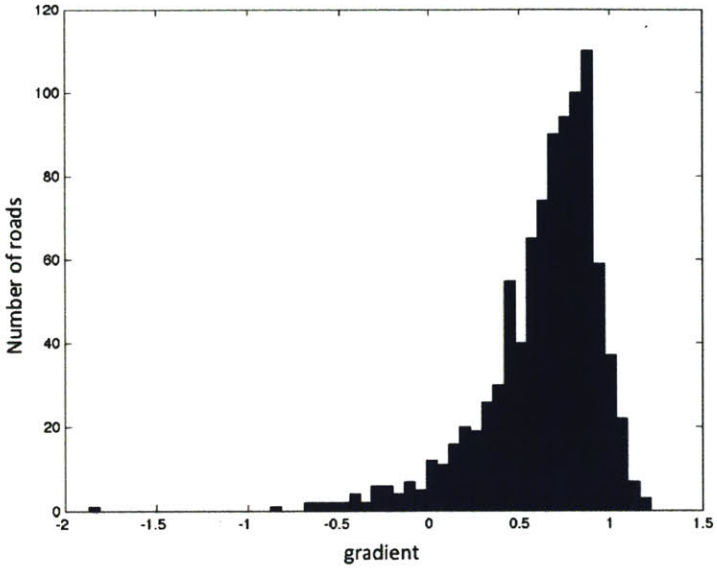


Figure 9-7: Histogram of the gradient of the line segment with the highest car density.

We also extracted the gradient of the line segment with the highest car density for each road. This is shown in the histogram in Figure 9-7. Notice that only a small percentage of the roads have line segments with negative gradients at high densities, which indicates that few roads in Singapore actually experience severe congestion, even if they do get slow-moving heavy traffic in the worst scenarios.

9.5 Synopsis

Understanding the relation between flow, density and speed of a road is a difficult yet important problem. The congestion effect of the road segment is modeled as the relationship between flow and delay. The model parameters were learned using the traffic volume data from loop detectors and GPS location and time data from a roving fleet of taxis. We demonstrate how empirical fundamental diagrams can be estimated for every road in the city without installing costly loop detectors at every intersection. Our method augments data collected from existing loop detectors with data from a fleet of taxis, which serve as dynamic probes into the traffic conditions. By integrating the two sources of data, we are able to accurately estimate traffic volumes and the critical point in fundamental diagrams, hence allowing us to detect congestion with high accuracy. Finally, we also show that roads may have operational states in addition to free flow and congestion, and our method is able to discover these multiple states from the empirical fundamental diagrams.

Chapter 10

From Theory to Practice: City-Scale Experiments

In this chapter, we present city-scale experiment using taxi trip data and inductive loop detector data in Singapore. Various analyses reveal the traffic and mobility patterns in the city. We learn the congestion model based on real data from a fleet of taxis and loop detectors. Using the learned street-level congestion model, we develop a congestion-aware traffic planning system that operates in one of two modes: (1) to achieve the social optimum with respect to travel time over all the drivers in the system or (2) to optimize individual travel times. We evaluate the performance of this system using 16,000 taxis trips and show that on average our approach improves the total travel time by 15%.

Section 10.1 shows city-scale applications of our traffic forecasting including traffic volume, hotspots, and origin/destination analysis. Section 10.2 compares the paths taken by real taxi drivers to the paths computed by our system (the social optimum paths) and the paths computed by a greedy path planning algorithm.

10.1 City-Scale Applications

Many interesting traffic and mobility analyses can be done using dynamic probes. In this section we give three examples of traffic analysis using dynamic probes: estimating traffic volume, estimating hotspots, and estimating the distribution of origins and destinations for traffic, which point to future directions and opportunities for using taxi probes to understand urban-scale mobility.

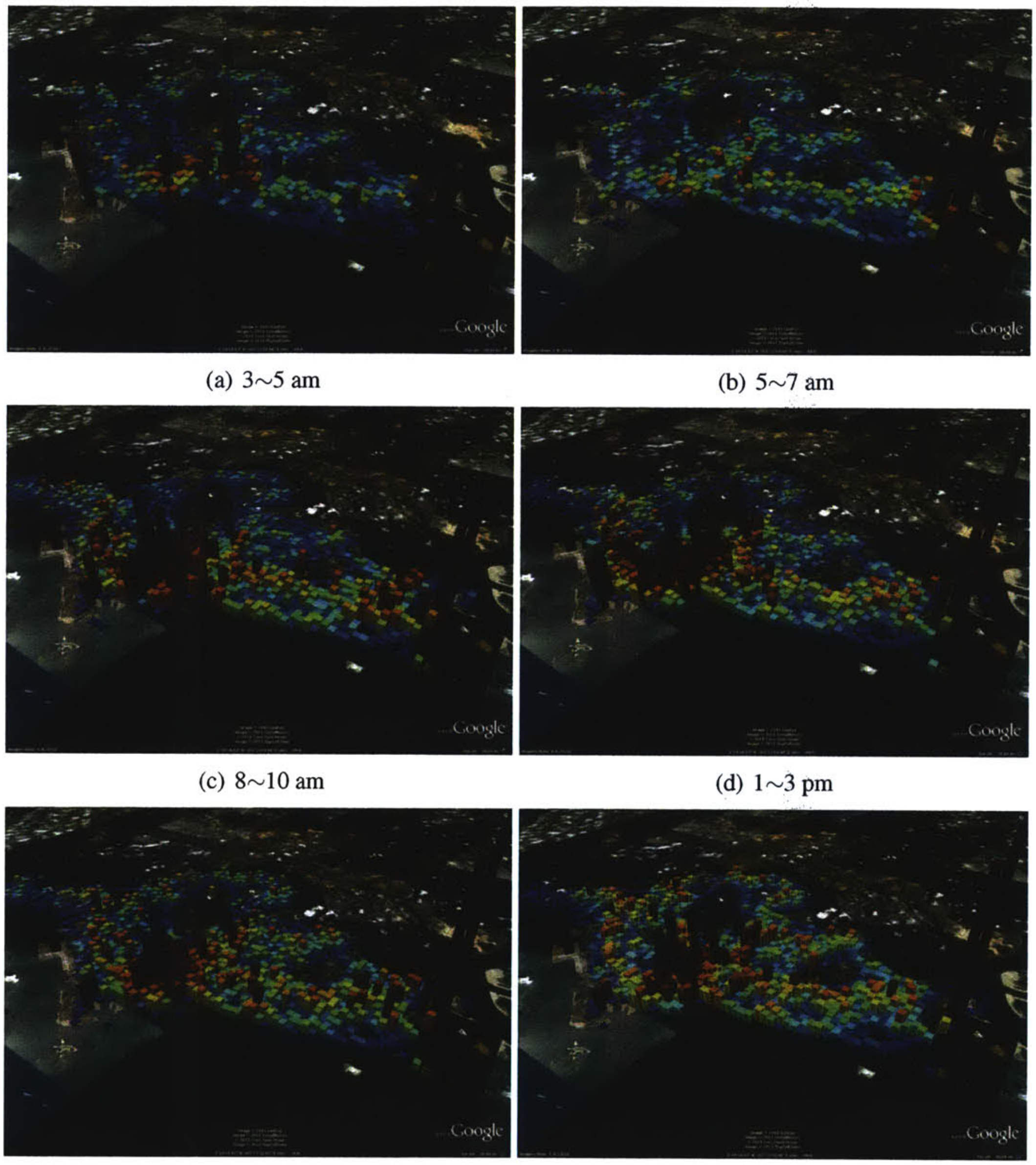
10.1.1 Volume

Figure 10-1 shows a snapshot of taxi volume for different hours of day on August 2nd (Monday) 2010 for Singapore. The volume is defined as the number of taxis observed in the square block over a given time and is plotted for a regional block size of 400 × 400 square meters and a time size of 2 hours. Color also encodes a qualitative measure of volume according the the color wavelength, with red denoting the highest volume and blue denoting the lowest volume. Each bar height measure the volume. While the volume distribution across the country changes according to different times of day, some parts of the country retain the larges fraction of the traffic volume. We discuss this phenomenon in the next section.

10.1.2 Hotspots

The red bars in Figure 10-1 indicate visually traffic hotspots. Some regions of the country remain hotspots regardless of the time of day (e.g. the Changi airport). Other regions (e.g. Orchard Rd) are hotspots during certain time periods. Figure 10-2(a) shows the top 9 hotspots, while Figure 10-2(b) shows the traffic variation during the day at each of these 9 hotspots (each location plotted as its own curve). We can observe different detailed volume variation over a day for those hotspots in Figure 10-2(b).

Hotspots such as 'A-bomb in Figure 10-2(a) show excess traffic volume in the early morning, and Hotspots such as 'F' show excess volume during the morning rush hour. 'B' shows excess traffic both in the morning and evening rush hours. 'C' (airport) show all-time high volume. Hotspots such as 'D', 'E' and much of northern area show that the traffic volume is relatively higher in the evening and early morning than morning rush hour.



(a) 3∼5 am

(b) 5∼7 am

(c) 8∼10 am

(d) 1∼3 pm

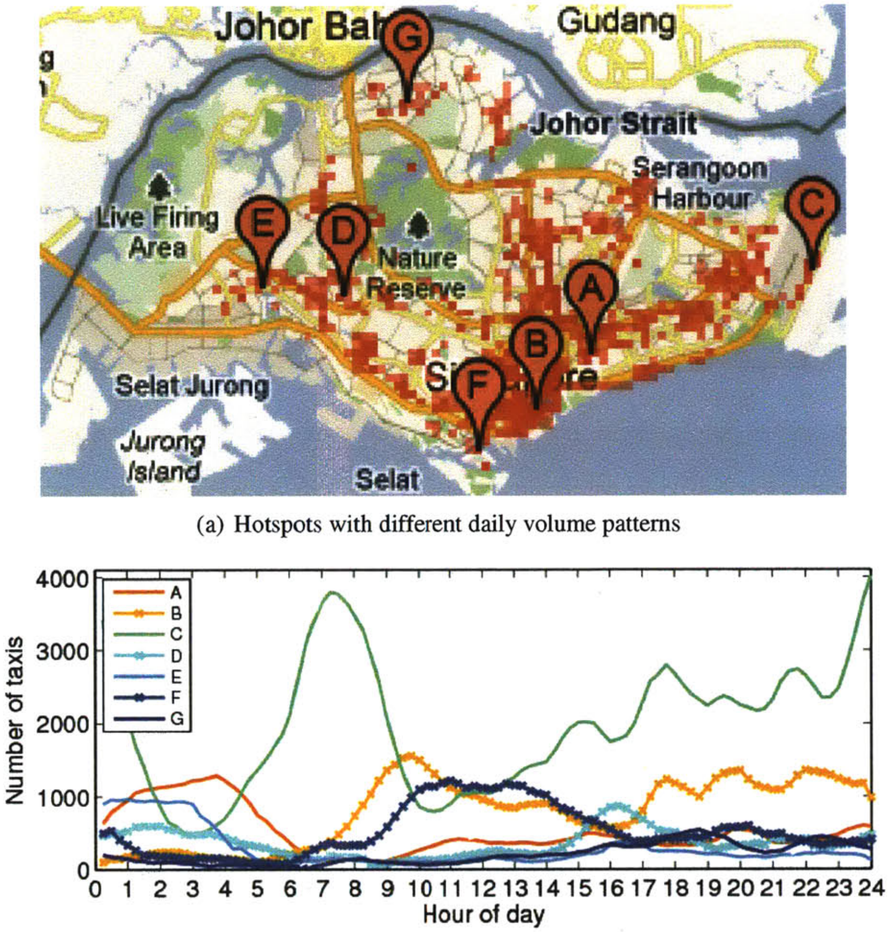
(e) 5∼7 pm

(f) 9∼11 pm

Figure 10-1: Snapshots of traffic volume measured by taxi probes for August 2nd 2010 (Monday) plotted on the map of Singapore. Bar height and color combine to encode traffic volumes. Note the large variation in volumes in different regions of the city.

10.1.3 Origins / Destinations

The streaming nature of the information from the taxi probes enable the aggregation of higher-order information for mobility analysis, for example where do trips originate and



(a) Hotspots with different daily volume patterns

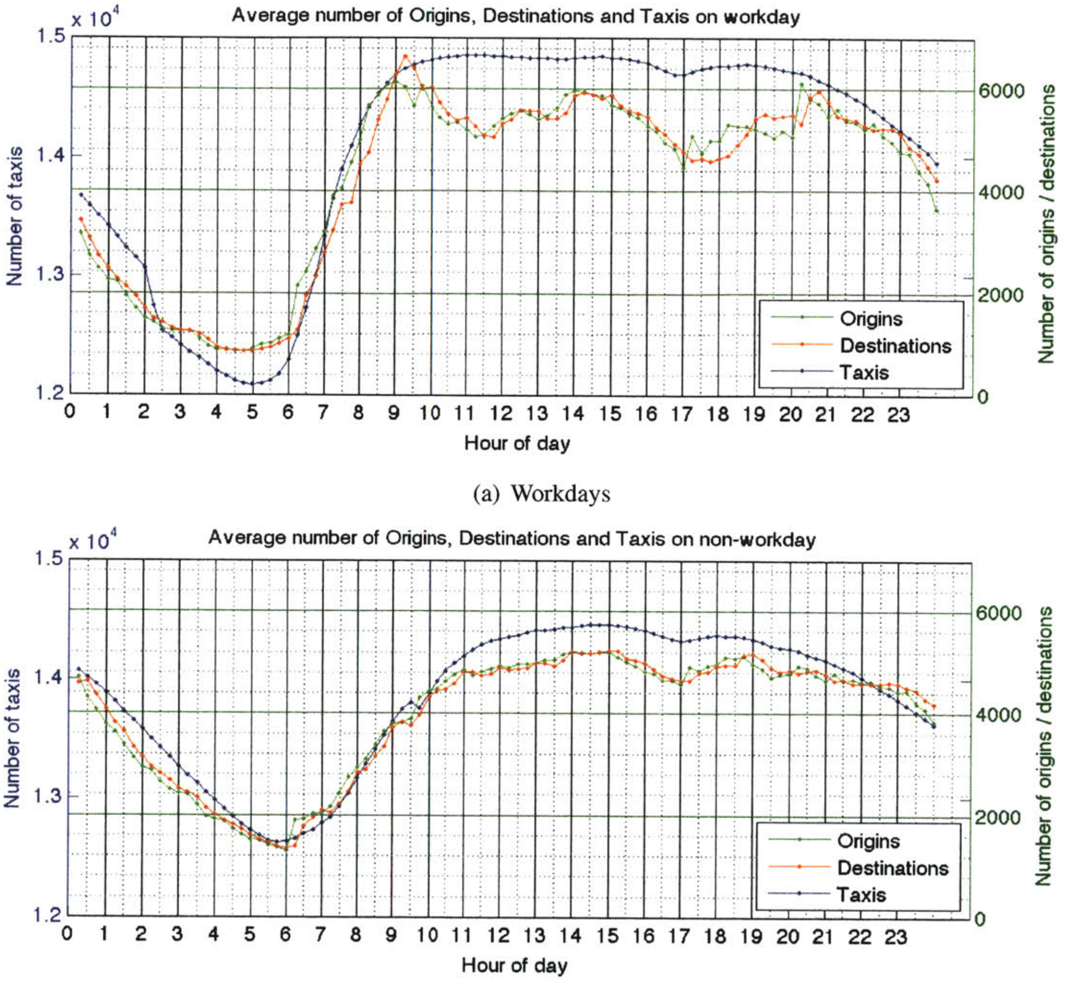
(b) Volume variation of hotspots over time

Figure 10-2: Hotspots according to time of day. The labels on the left map show the locations of the hotdpots. The curves show volume vs time of day for each of the 9 identified hot spots.

end and which trajectories were followed. In this section we give results on how dynamic probes can be used to analyze origins and destinations. In our future work we will describe how we can use dynamic probes to infer general traffic mobility patterns.

Number of Taxi Trips

A total of approximately 12 million trips were extracted from the taxi data, and the average number of trips starting (origins) and ending (destinations) at different times-of-day are shown in Figure 10-3, together with the number of taxis plying the road.



(a) Workdays

(b) Non-workdays

Figure 10-3: Number of taxis and trips (in terms of origins and destinations) at different times of days, on both workdays and non-workdays.

The number of taxis decreases from 8pm to a daily low at 5am, before drastically increasing through the morning rush hours till 10am. The supply of taxis on the roads then remains nearly constant until the 8pm in the evening. The slight dip in taxi numbers at 5pm is possibly due to shift changes.

The pattern for the demand of taxis (as represented by the number of origins / destinations) is similar to that of the supply of taxis, where fewer trips are taken between 8pm to 5am, before again increasing till 10am. On workdays, however, the peak demand is experienced at 9am. Even though the demand decreases after 9am, taxi drivers have started their own working hours, and continue to ply the roads, resulting in an over-supply during this period.

Distribution of Origins and Destinations

Each origin and destination from all 12 million trips were classified as belonging to one of the 28 regions. For each fifteen minute block starting from midnight on workdays, we calculated the empirical distributions of the origins and destinations over the 28 regions. This procedure was also repeated for trips on non-workdays. In total, 384 empirical distributions were obtained (one distribution for origins on workdays, origins on non-workdays, destinations on workdays, and destinations on non-workdays for each of the 96 fifteen minute blocks in a day).

The randomness of each of these distributions were measured using *perplexity*.

Definition 4. *Perplexity for a distribution p is defined as* $2^{H(p)} = 2^{-\sum_z p(z)\log_2 p(z)}$, *where* $H(p) = -\sum_z p(z)\log_2 p(z)$ *is the entropy of the distribution.*

The perplexity has a natural interpretation for the OD distributions: if a trip has equal probability of starting in any region, then the perplexity of the empirical distributions of origins is exactly equal to 28, the number of regions. On the other extreme, if all trips begin in the same region, then the perplexity is equal to 1. A distribution with a perplexity value of X is as random as a uniform distribution over X regions.

The lines of marker 'O' in Figure 10-4 show the perplexities of the empirical distributions of origins and destinations at different times during workdays. The peak in the perplexities of origins is found at the morning rush hours as passengers leave their homes across Singapore. Distribution of origins stabilizes in the afternoon, before undergoing a drop from 10pm to the trough at 3am. During this period, trips largely originate from a small number of regions near the city centre that are dominated by offices and popular shopping malls, indicating that passengers are reversing their morning trips and headed home.

Conversely, a trough in the perplexities of destinations occurs at the morning rush hours,

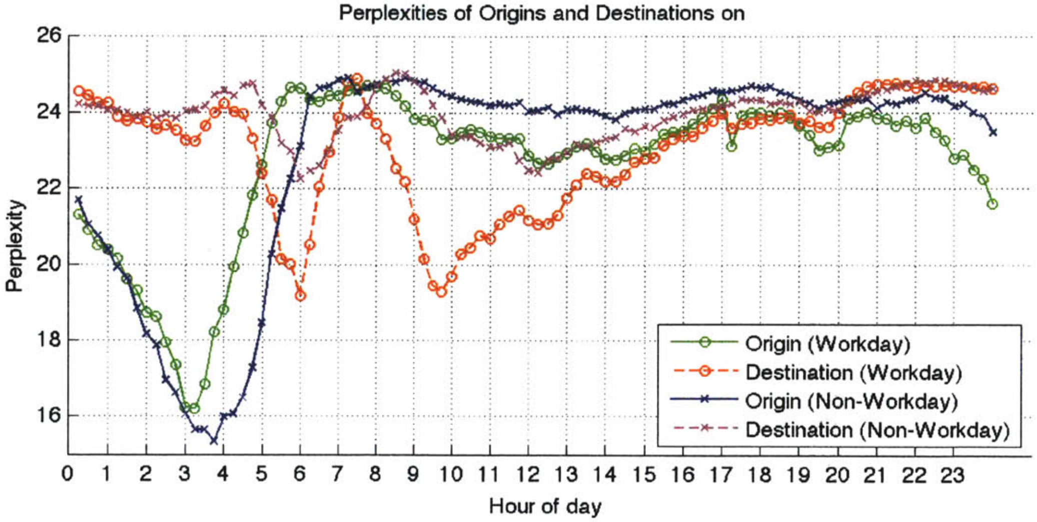


Figure 10-4: Perplexities of empirical distributions of origins and destinations

when passengers are traveling to the small number of city regions. Throughout the rest of the day, the perplexity increases as passengers travel to more diverse regions. Between 8pm - 12pm, perplexity of destinations reaches its nightly peak as passengers head to their homes in a large number of residential regions. A second trough occurs at 6am – this phenomenon will be explained below.

Similar patterns can be seen on non-workdays as well, as shown by the lines of marker 'X' in Figure 10-4. In contrast to workdays, however, perplexities on non-workdays tend to be higher, indicating that the population's movements are less synchronized. Furthermore, the trough in perplexities of destinations, has shifted from 9:45am on workdays to 12:15pm on non-workdays. This suggests that a shift in the behavior of Singapore residents has occurred between workdays and non-workdays.

Unfortunately, perplexities are coarse-grained summary statistics that fail to properly account for the richness of behavior encoded in the full distributions. To better understand the evolution of OD distributions across time, we examined the probabilities of origins and destinations occurring in each region.

We have highlighted 6 regions of interest for discussion, 5 of which fall within or near the city center. The labels we have used for naming the regions are purely descriptive, and do not necessarily correspond to any official naming or demarcations. Nevertheless, a Singapore resident should be able to readily identify the regions and common activities associated with them.

The 6 regions of interest (as indicated by the corresponding bright colors in Figure 7-4) are:

1. Airport (Black): The easternmost region of Singapore holds Changi Airport, by far the most important civilian airport.

2. Geylang (Red): Although a largely residential area, Geylang is also well- known for some of its nocturnal activities.

3. Orchard (Green): Singapore's famous shopping strip, but also houses a substantial number of offices.

4. CBD (Blue): The Central Business District is dominated by high-rise offices, but also caters to party-goers with its pubs and clubs.

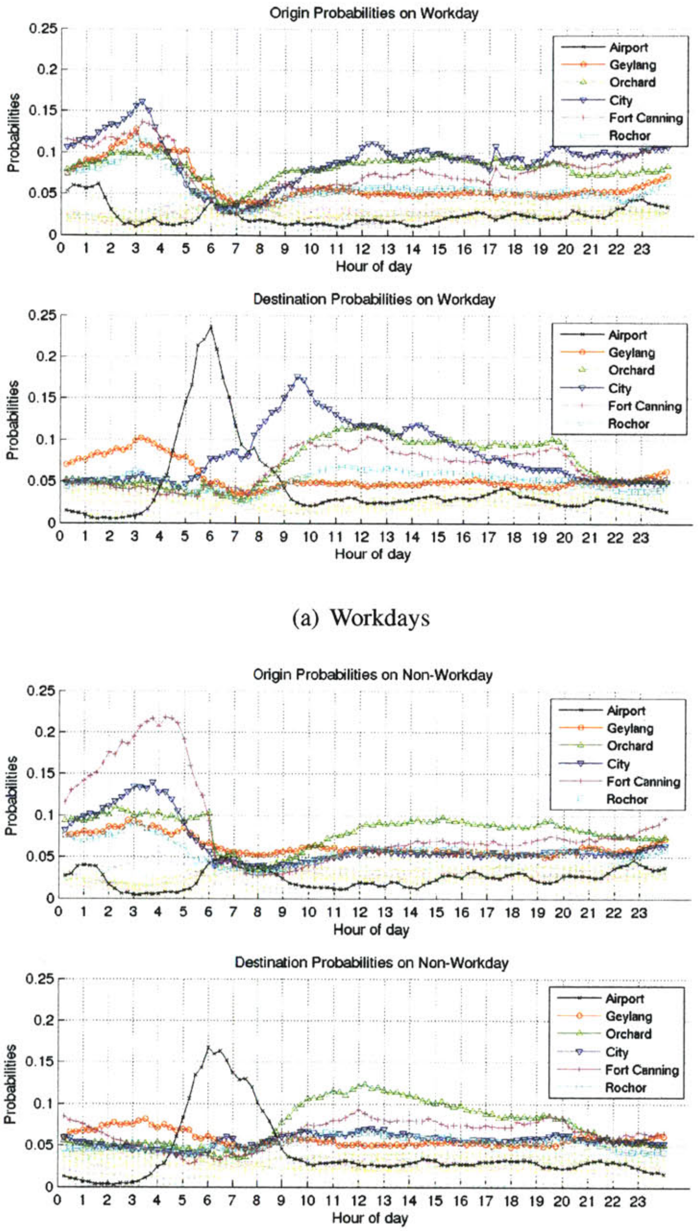
5. Fort Canning (Magenta): Some of Singapore's most popular clubs are found here. Being adjacent to Orchard and CBD, there are also shopping malls and offices in the area as well.

6. Rochor (Cyan): At the fringes of the city area, Rochor is home to a few high-rise offices and shopping malls.

Figure 10-5 shows the probabilities of trips originating and terminating in the 28 Voronoi regions. The 6 regions of interest are highlighted while the remaining 22 (mainly residential) regions are faded into the background. We discuss some of the patterns that emerge from these plots.

The city regions (CBD, Orchard, Fort Canning, and to a smaller extent, Rochor and Geylang) are the most popular regions for taxi trips to begin and end at most times of the days. The prominence of CBD as a destination from 8am to 10am reflects the arrival of people at their offices. As the day wears on, the CBD is then overtaken by other city areas such as Orchard and Fort Canning, where more leisure activities can be found.

One exception to the dominance of the city regions is the morning hours from 6am to 8am. During this time, one is more likely to find trips originating from residential areas as people leave their homes to begin their day. Conversely, the popularity of city regions



(a) Workdays

(b) Non-workdays

Figure 10-5: Probabilities of empirical distributions of origins and destinations. Taxi trips' origins and destinations are mostly concentrated in the city regions, except for the morning rush hours where origins are dominated by residential regions. The airport is also a popular destination in the early morning hours.

as origins increases during the evening, since taxi passengers are now leaving the city for home. The same effect is observed with the destination probabilities when the city regions become unpopular after 8pm.

Another pattern that can be clearly seen from Figure 10-5 is the popularity of the Airport region as a destination in the early morning hours. We hypothesize that a large number of flights are scheduled to leave Singapore in the morning. This is further compounded by the lack of public trains in the early hours, leaving taxis as the major means of public transportation for air travellers. This spike in the popularity of the Airport region also accounts for the trough in perplexities of destinations at 6am seen in Figure 10-4.

The patterns on non-workdays (Figure 10-5(a)) are similar to the patterns on workdays (Figure 10-5(b)). A notable exception is the CBD, whose popularity as a destination drops drastically due to fewer people returning to offices on non-workdays.

On the other hand, during the late night hours 3am - 5am on non-workdays, trips are likely to originate from the Fort Canning region. This is likely to be due to the presence of nightspots that are frequented only on non-workdays when one does not have to report to work early the next morning.

O-D Relationship

In addition to studying origins and destinations separately, we also investigated the relationship between the two. Specifically, we examined if the knowledge of a trip's origin could provide us with information about its destination, and vice versa. Such questions would be of particular interest to taxi drivers, whose expectations of their next task could be altered according to the location where the passenger is picked up. OD transitions are also important for transport planners as a first step towards understanding the load on the traffic network.

In Figure 10-6, we compare the perplexities of origins and destinations with the *conditional perplexities* of origins given destinations and of destinations given origins respectively.

Definition 5. *The conditional perplexity of origins given destinations is defined as* $2^{H(O|D)}$, *where* $H(O|D) = -\sum_o \sum_d p(o,d) \log_2 p(o|d)$ *is the conditional entropy of origins given* *destinations.*

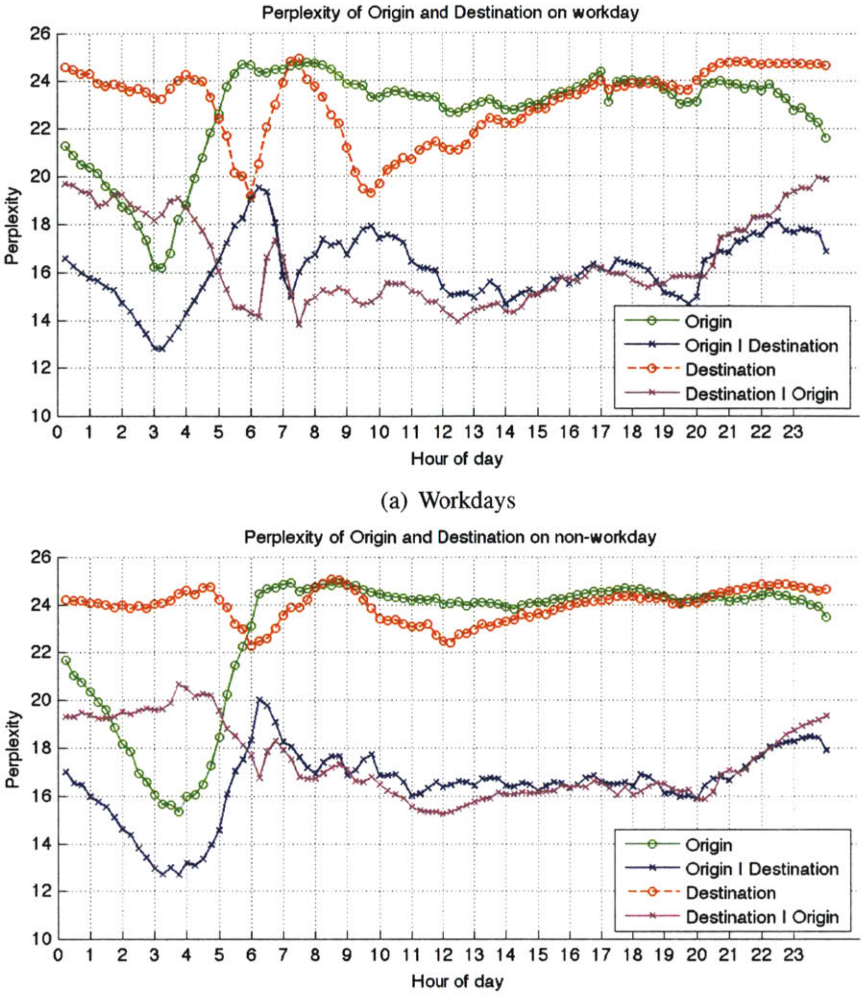
The conditional perplexity of destinations given origins is similarly defined.

The conditional perplexity of origins given destinations is the randomness or uncertainty about origins that remains after the destination has been made known. Thus, if the origins and destinations were independent, then we gain no information about origins through the knowledge of destinations, and the conditional perplexity is equals to the perplexity itself. Such a situation may arise if all trips converged on a single region (so the destination provides no additional information about the origin), or if all trips originated from the same region (so the origin provides no additional information about the destination). On the other hand, if the two are perfectly correlated, then knowledge of destinations leaves no uncertainty about the origins, so the conditional perplexity is 0. The difference between the perplexities and the conditional perplexities quantifies the information that one provides about the other.

As Figure 10-6 shows, knowledge about destinations reduces the perplexities of origins by 3.5-9.5 regions on workdays and 2.5-8.5 regions on non-workdays. Knowledge about origins reduces the perplexities of destinations by 4.5-11 regions on workdays and 4-8.5 regions on non-workdays. The differences between perplexities and conditional perplexities are, however, not uniform across different times-of-day and workday / non-workday. Figure 10-7 shows the variation of the ratios of perplexities to conditional perplexities across the day.

The mutual information $I(O;D) = H(O) - H(O|D) = H(D) - H(D|O)$ is a symmetric measure of dependence between the origins and destinations. The ratio between the perplexities and conditional perplexities works out to be exactly $2^{I(O;D)}$, which is shown in Figure 10-7. We see that mutual information tends to be higher in the day than at night. Furthermore, a spike between 7am to 8am, followed by a depression at 10am, is observed on weekdays. To explain these observations, we examined the proportion of intra-regional trips, i.e. trips which originated and terminated within the same region.

As shown in Figure 10-8, the proportion of intra-regional trips is correlated with the mutual information between origins and destinations. At 7am-8am on workdays, many taxi trips are made within the regions around the perimeter of Singapore. We believe that this is



(a) Workdays

(b) Non-workdays

Figure 10-6: Conditional perplexities of origins given destinations, and vice versa.

caused by people traveling to work locations near their homes, and hence knowledge of the trip origin provides useful information on the probably nearby destination. This behavior is then overtaken by the movement of office workers into the city regions at 10am; since all trips likely end in the city, the knowledge of the trip's origin provides little additional information.

The proportion of intra-regional trips increases after 10am as people do not move far from their workplace. This allows us to again make better guesses about the destination if the origin is known. After work, however, trips are taken out of the city into the residential

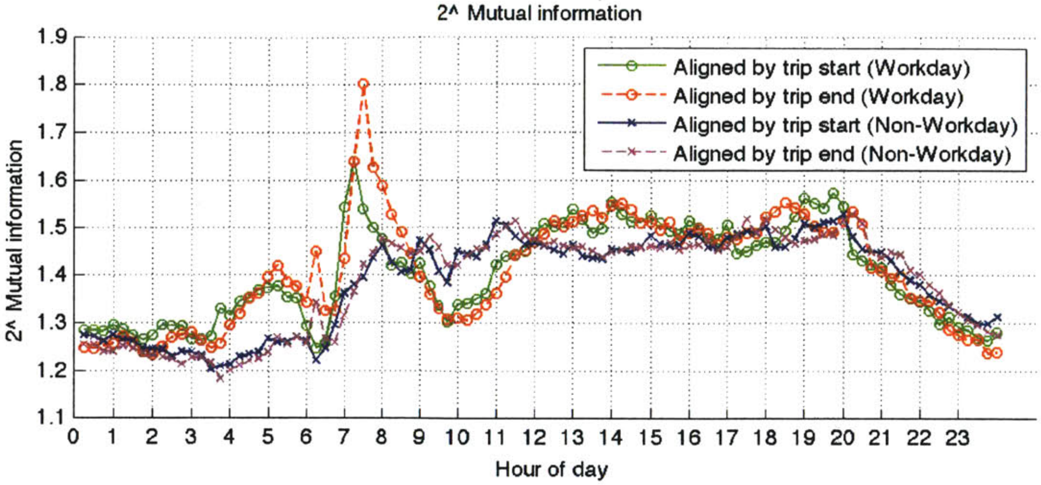


Figure 10-7: Mutual information between origins and destinations. A clear spike occurs between 7am to 8am on workdays, caused by an increase in the probability of intra-regional trips (see Figure 10-8).

areas; since all trips likely originated from then city, the knowledge of the trip's destination provides little additional information. The low proportion of intra-regional trips (and low mutual information) then persists until the next morning.

Similar patterns are observed on non-workdays as people make short intra- regional trips in the day but disperse at night. The spike and depression are not prominent, however, since there is less work-related movement.

10.2 Experimental Comparative Study on Travel Time

In this section, we quantify empirically the travel time performance of a fleet of taxis for the case when the taxis use the social optimum path computation algorithm from Section 7.1. We compare travel times as recorded by the real taxi data to the computed and-simulated greedy-optimal paths and socially optimal paths.

10.2.1 Approach

We begin by computing the total traffic for each road segment for each 15-minute time slot using the methods described in Section 8.3. We identify the taxi trips and re-route them us-

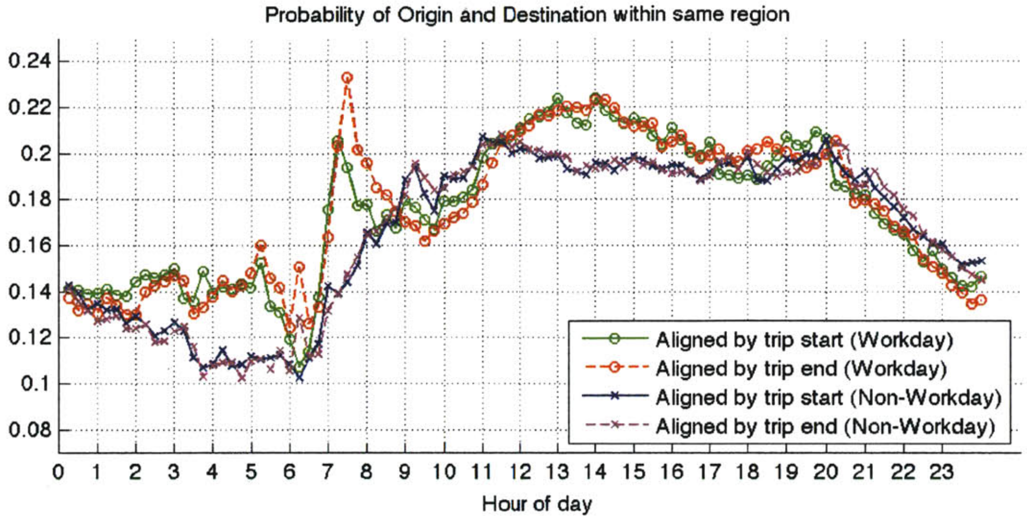


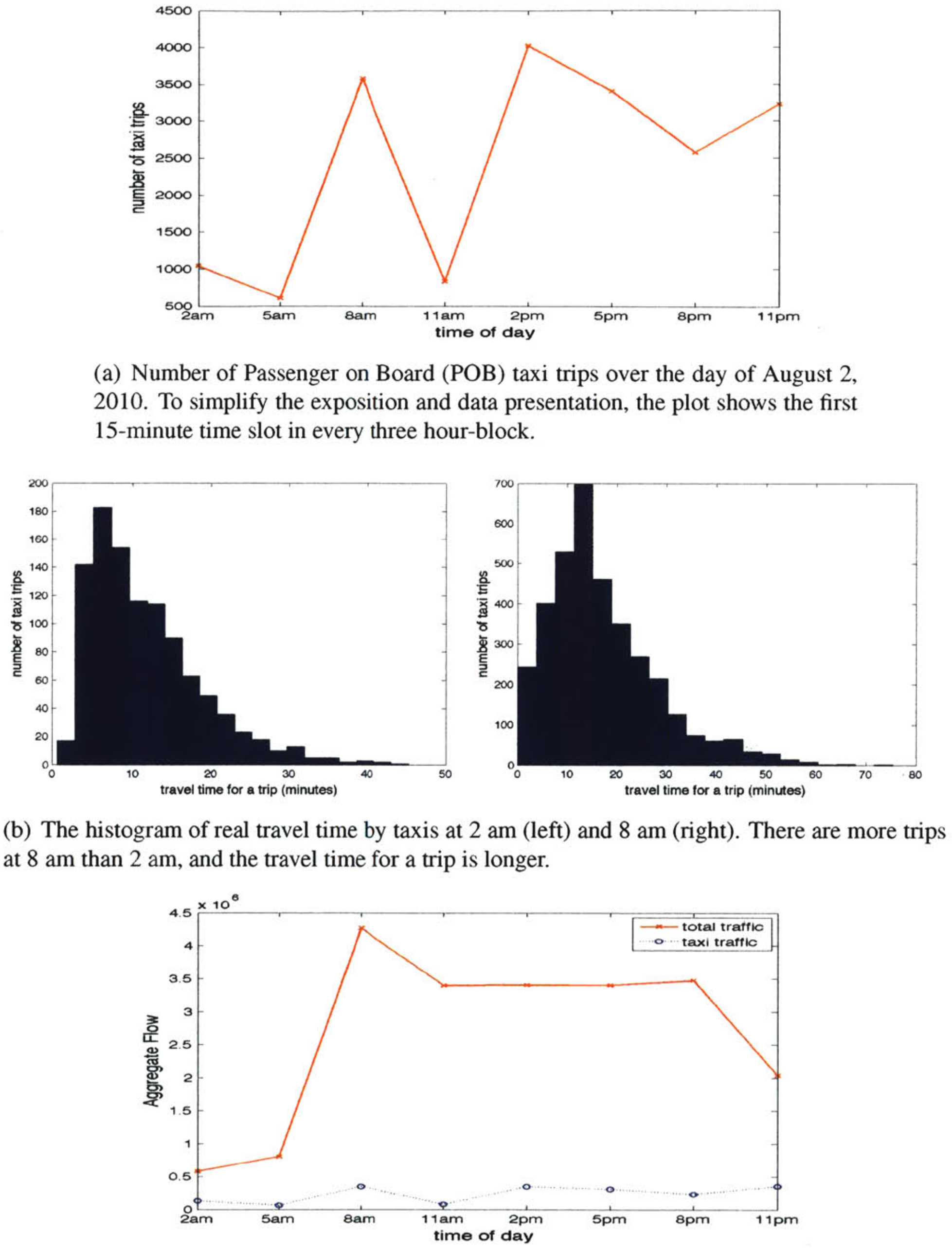
Figure 10-8: Probabilities of a trip originating and terminating within the same region, at different times of day, on both workdays and non-workdays. The spike between 7am to 8am on workdays corresponds to the increased mutual information between origins and destinations at the same time.

ing the algorithm in [84] based on the flow-delay function that we learned in Section 10.2.2.

Computing Taxi Traffic Flow

For each 15-minute time slot, we select only the taxi trips flagged as Passenger on Board (POB) trips in the taxi data set. The GPS location where the state of the taxi changes to POB is considered the start of the trip. The GPS location where the POB state is reset to a different staus is considered the destination of the trip. Figures 10-9(a) and 10-9(b) show the number of POB taxi trips and the histogram of travel times of the POB taxi trips, respectively. The GPS points recorded by a taxi between the origin and destination of a trip are matched to the Singapore map using an algorithm similar to [99]. Using this map matched sequence of roads, we find the flow generated by POB taxis for each road segment. The POB taxi flow as well as the total traffic flow for overall road network is shown in 10-9(c).

We compute the total real travel time of all POB trips in a 15-minute time slot, and compare its cost to the greedy-optimal paths and the social optimal paths for the simulated case where the POB taxis follow the computed paths.



(a) Number of Passenger on Board (POB) taxi trips over the day of August 2, 2010. To simplify the exposition and data presentation, the plot shows the first 15-minute time slot in every three hour-block.

(b) The histogram of real travel time by taxis at 2 am (left) and 8 am (right). There are more trips at 8 am than 2 am, and the travel time for a trip is longer.

(c) Total traffic and taxi traffic over the day of August 2, 2010. To simplify the exposition and data presentation, the plot shows the first 15-minute time slot in every three hour-block. Taxi traffic accounts for about 20% of the total traffic at 11 pm and 2 am and only about 2% at 11 am. For the rest of the time of the day, the portion of taxi traffic is about 10% of the total traffic.

Figure 10-9: Statistics of Selected Taxi Trips

Computation for New Assignment

The *underlying traffic flow* is the traffic that we do not control. The underlying traffic flow consists of all non-taxi traffic and non-POB taxi trips. The underlying traffic is computed by subtracting the POB taxi traffic flow from the total traffic flow.

The greedy-optimal path is defined as the minimum expected time path. The greedy-optimal path will have short travel time for single user queries; however, if multiple users are directed along the same greedy path, the performance of the path degrades.

We compute the path sets of size K used in the multi-user routing algorithm in Section 7.1 for each POB taxi trip's origin and destination pair. First, the real taxi path for each O-D pair is included to its path set, P. Then, the other $K-1$ paths are found by a standard shortest-path algorithm such as Dijkstra's algorithm using the mean travel time statistics of road segments for the corresponding 15-minute time slot. The details of the method we used to find a reasonably disjoint set of $K-1$ paths are as follows: We define B to be a set of road segments that will not be used for finding an O-to-D path. First, we set B empty and find the minimum expected time path using Dijkstra's algorithm. This path is added to P. Second, we randomly select a predefined number of road segments from the found path, and add these road segments to B. We find the O-to-D path that minimizes the expected time not using any road segment in B. The found path is added to P. We do this process until we have total K paths in P. An example of 5 paths found using this method for an O-D pair is shows in Figure 7-1.

10.2.2 Link Performance Function Estimation

Link Performance Function

The Bureau of Public Roads (BPR) developed a link congestion (or flow-delay, or link performance) function, which we will term $t_a(f_a)$ [103]:

$$t_a(f_a) = T_a^0 \left(1 + \alpha \left(\frac{f_a}{C_a} \right)^{\beta} \right), \tag{10.1}$$

where T_a^0 is free flow travel time on link a per unit of time; f_a is flow attempting to use link a per unit of time; C_a is capacity of link a per unit of time; and $t_a(f_a)$ is the average travel time for a vehicle on link a.

The BPR function is frequently used for computing an optimum traffic assignment [30, 77, 84]. Values for α and β are empirically measured from data. They may be different for different type of roads, although typical values for α and β are 0.15 and 4, respectively, based on the empirical data on highways. Our goal is to find the best empirical model describing the data using flow and travel time data for each road segment in Singapore, given our two data sources.

Link Performance Function Estimation using Flow Estimation

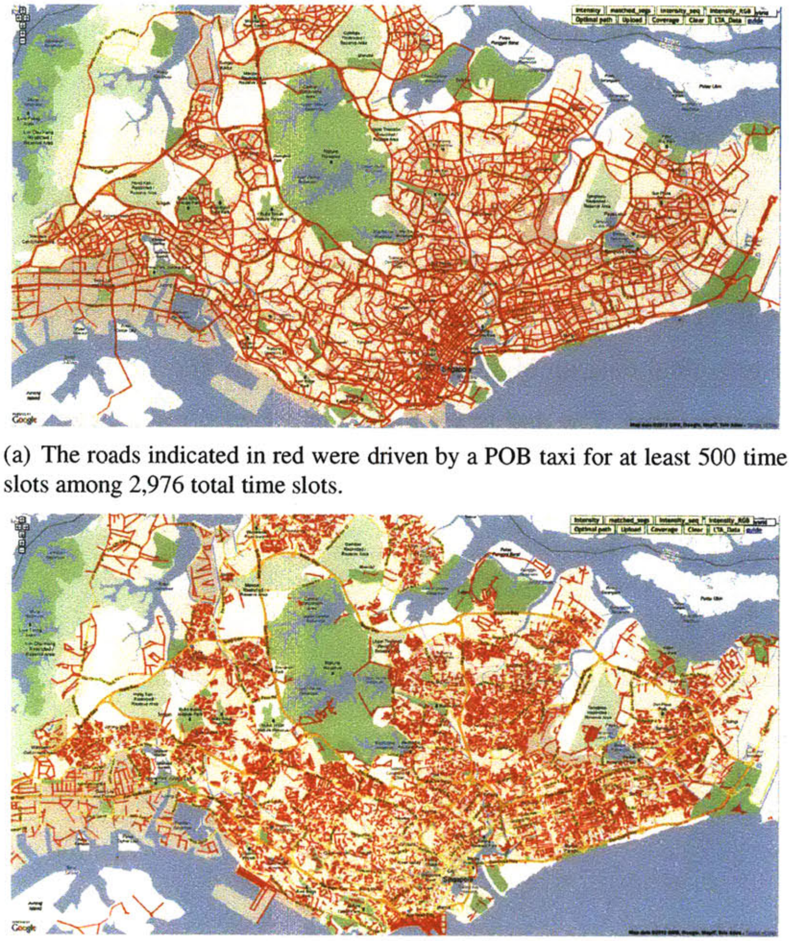
Eq (10.1) can be rewritten as:

$$t_a(f_a) = p_{1a} + p_{2a} f_a{}^{\beta}, \tag{10.2}$$

where $p_{1a} = T_a^0$, and $p_{2a} = \frac{T_a^0}{C_a{}^{\beta}}$. Our goal is to find the best β that governs the link performance function, and find p_{1a} and p_{2a} for each link a that fit the empirical data.

We first select roads with enough data points to learn the flow-delay curve. The roads where more than 500 time slots (among the total 2,976 time slots) have at least one taxi data point are selected. We estimate the link performance function for these selected roads (approximately 38,000 roads, 49 % of overall 77,000 roads) as shown in Figure 10-10(a). Second, for the roads rarely driven (Figure 10-10(b)), we calculated T_a^0 using the speed limit. We set C_a as 100, the most frequently observed C_a in the first step.

We find the best parameter that fits all the data points in 2,976 time slots in August, 2010, one data point (flow and delay) per 15 minute slot. We fit the data using β = 2, 3, and 4 and found that $\beta = 2$ maximizes the R squared value. Figure 10-11 shows the data fit using the function with β = 2.



(a) The roads indicated in red were driven by a POB taxi for at least 500 time slots among 2,976 total time slots.

(b) All the other roads not in (a)

Figure 10-10: Road segments with taxi data

10.2.3 Algorithm Implementation

Our data set includes several thousand simultaneous trips that need to be planned as shown in 10-9(a). Using a modest computing platform (4 2GH CPU cores and 8 GB main memory), we select a random subset of size 100 from the total trips to plan and compute the social optimum for a predefined time interval. We save the computation result and iterate this procedure with other random subsets.

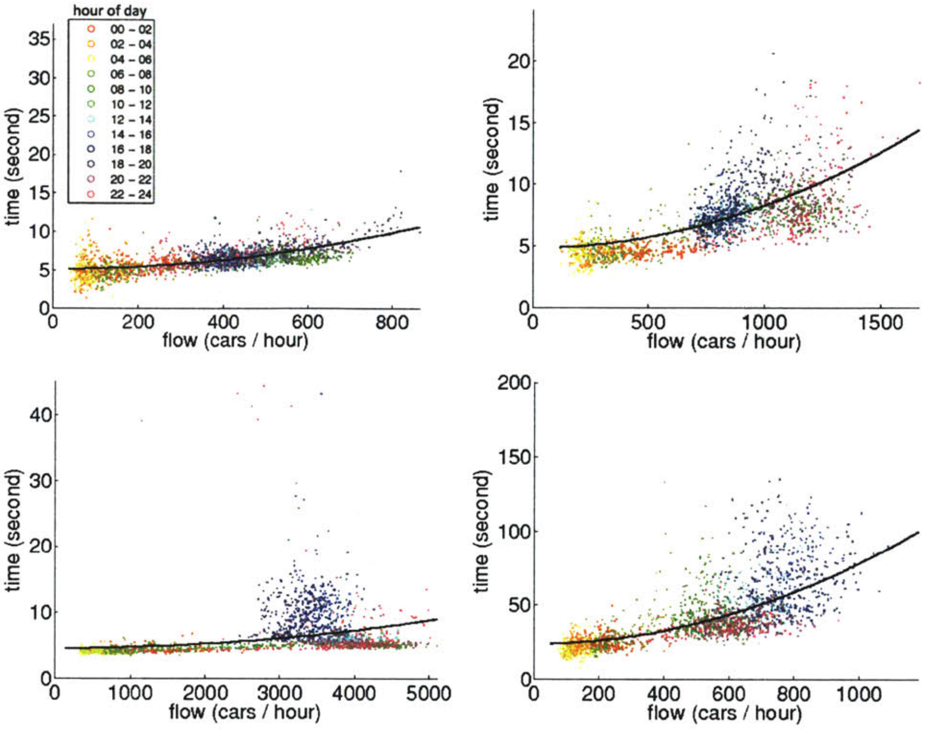


Figure 10-11: Goodness of fit for estimated link performance functions. The data points are color-coded by the time of day as indicated in the legend. A data point represents flow and speed information gathered by loop detectors and taxis for each 15-minute time block in August 2010.

10.2.4 Results

In this section we describe the results of our experiments for all the POB taxi trips for August 2, 2010 (Monday). We used 5 paths for each O-D pair. One path was chosen as the real path driven by the taxi; 4 paths were computed using the mean travel time of road segments calculated from taxi data.

The travel time comparison between real taxi paths, greedy paths, and social optimum paths of different times of the day is shown in Figure 10-13. For example, the total travel time for 3,400 trips during 5 pm ~ 5:15 pm on August 2, 2010 is 769 hours, which is 13.58 minutes per trip on average. The total travel time for the total 3,400 trips computed using our congestion model is 728 hours, or 12.86 minutes per trip on average. At the social optimum, the total travel time is 641 hours, which is 11.32 minutes per trip on average.

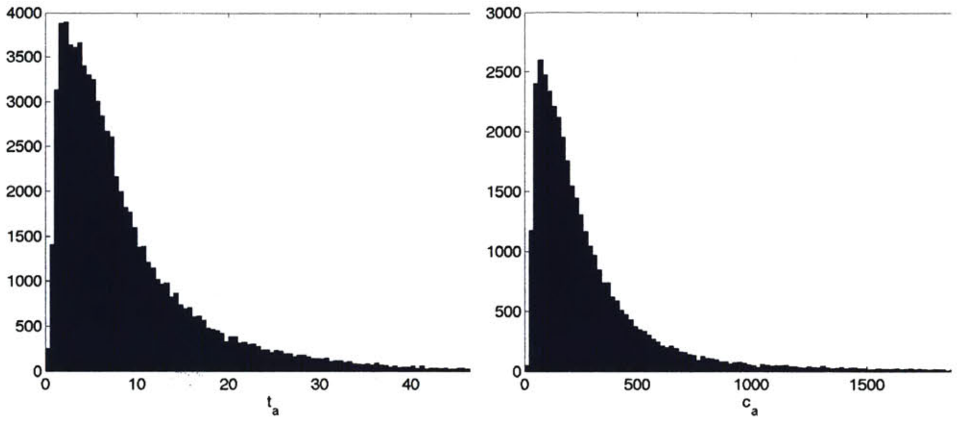


Figure 10-12: Histogram of T_a^0 (left) and C_a (right)

```
Algorithm 18: Compute-Social-Optimum
Data: underlying traffic, O-D pairs and path sets for all the trips to be planned;
Result: socially optimal paths;
Load underlying traffic;
Initialize link flows using the underlying traffic;
for a random subset of the total trips do
    Load saved underlying traffic and path selection probability for each driver;
    Set link flows using the underlying traffic.;
    Run Algorithm 14 for a predefined time interval;
    Save the underlying traffic and path selection probability for each driver;
end
```

Thus, the congestion-aware social-optimum algorithm saves on average 15% of the total travel time (approximately 2 minutes per trip on average). The total saved time for the taxi company for the day's operation would have been 395 days if the taxis followed our social optimum assignment. Note that this saving is only for POB taxi trips, which range from 500 to 4,500 for each 15-minute time slot depending on the time of day as shown in Figure 10-9(a).

To see how much we can save depending on the number of agents participating in the social optimum assignment, we conducted an experiment using only sub-sampled taxis for social optimum assignment. The savings for taxis are calculated by observing the difference between the estimated travel time of those taxis for the real paths and the estimated travel time in social optimum assignment. To calculate the savings for the total traffic we find the

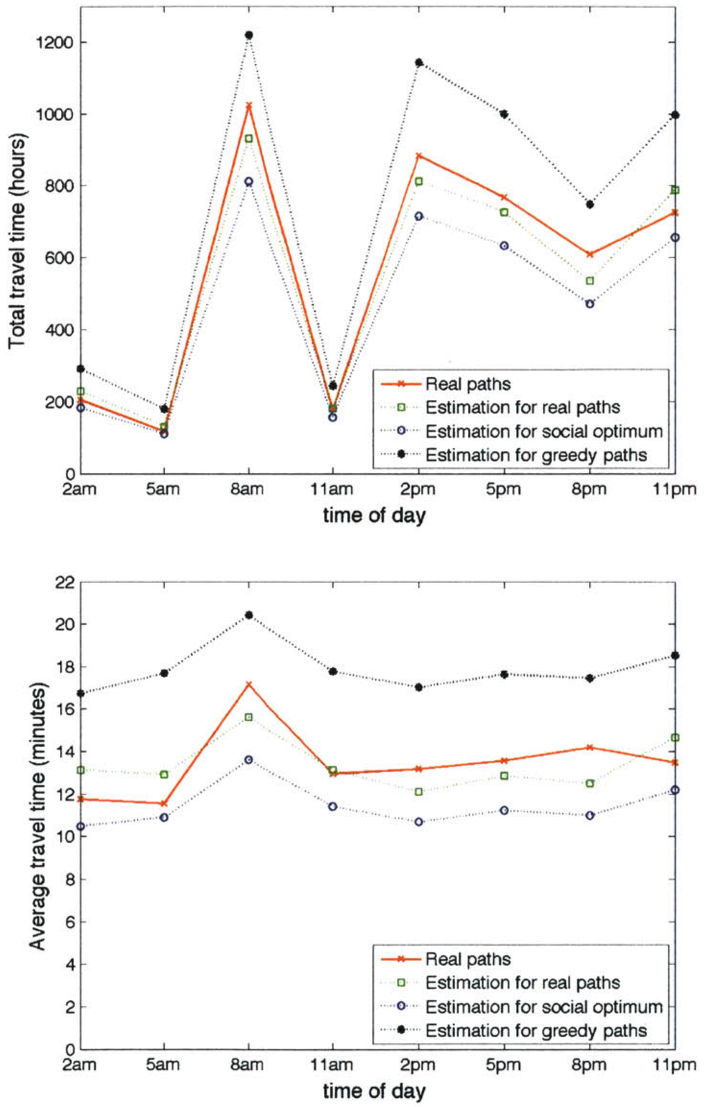


Figure 10-13: Total cost (top) and average cost (bottom) per trip for real taxi trips, greedy-optimal paths and socially optimal paths over time of day. The plots are computed using the real taxi data for Monday, August 2, 2010. To simplify the exposition and data presentation, the plot shows the first 15-minute time slot in every three hour-block. Our estimates using the congestion model for the real taxi trips (green) are close to the cost experienced by the taxis (red). The socially optimal assignment (blue) has about 15% reduction of the total cost. This is equivalent to 200 hours of travel time savings for 3,600 taxis from 8 am to 8:15 am. The greedy assignment based on the historical information produces worse (longer) paths than the real taxi trips. This shows that the single user traffic planning algorithms can be detrimental when they are used simultaneously by many users.

flow of edges and the travel time for each edge using the learned link performance function. Then, we find the total cost. We find the savings for general traffic by subtracting this from the original total travel time of general traffic. Figure 10-15 shows the effect of number of taxis on the savings. The time saved for participating taxis increases as more taxis get involved with social optimum assignment, and its growth rate decreases slightly as more taxis are involved. The savings for total traffic (including taxis) also increases. This shows that taxis' social optimum assignment considers the general traffic and that the general traffic gains advantage from these participating taxis. The savings for general traffic are 2.12 % of the travel time for the general traffic when 3,300 taxis follow our social optimum assignment. Thus, we can expect more savings for total traffic when the number of cars participating in our social optimum assignment grows.

We can clearly observe that the saving for general traffic increases as more taxis are used. This is because our social optimum assignment algorithm not only considers the cost of taxi but also the general traffic's cost. Thus, as we have additional drivers join the social optimum assignment, we can expect that the total cost only decreases.

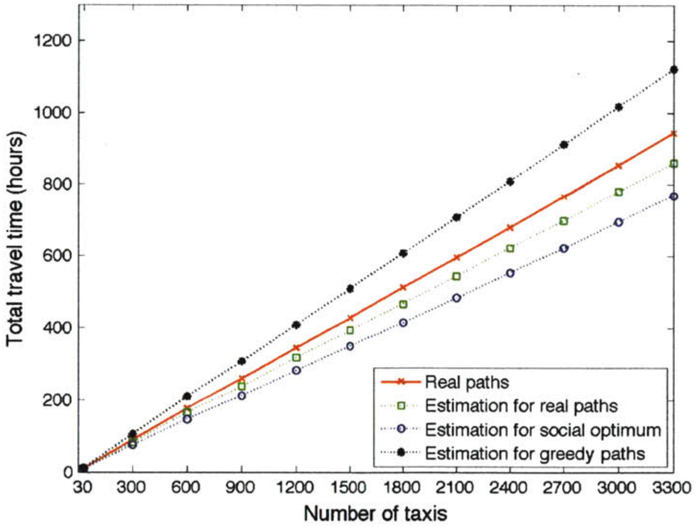


Figure 10-14: The travel time for real taxi trips, greedy-optimal paths, and socially optimal paths over the number of taxis used. We subsampled the total 3,600 POB taxis that start their trips between 08:00~08:15am on August 2, 2010. The x axis denotes the number of taxis used for our social optimum assignment. The y axis indicates the total travel time for the number of taxis used.

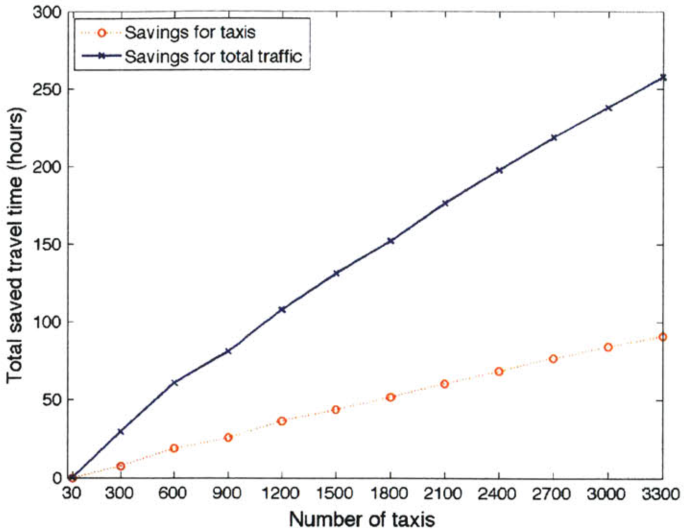


Figure 10-15: The savings for travel time for social optimum assignment of sampled taxis. We subsampled the total 3,600 POB taxis that start their trips between 08:00∼08:15am on August 2, 2010. The x axis denotes the number of taxis participating in our social optimum assignment. The y axis indicates the number of hours saved. The savings for participating taxis are shown as 'Savings for taxis' and the savings for total traffic are shown as 'Savings for total traffic' based on the estimated travel time at social optimum assignment and estimated travel time for the real assignment.

Figure 10-16 shows the trip time histogram for 8:00∼8:15 am on August 2, 2010. It shows that the travel times for trips are reduced when we assign the taxis through social optimum paths. The total traffic flow and the taxi flow for the same time slot are visualized in Figure 10-17. We can clearly see that the social optimum planning system distributes taxi traffic in a way that avoids congested road segments, whereas the greedy solution generates more congestion.

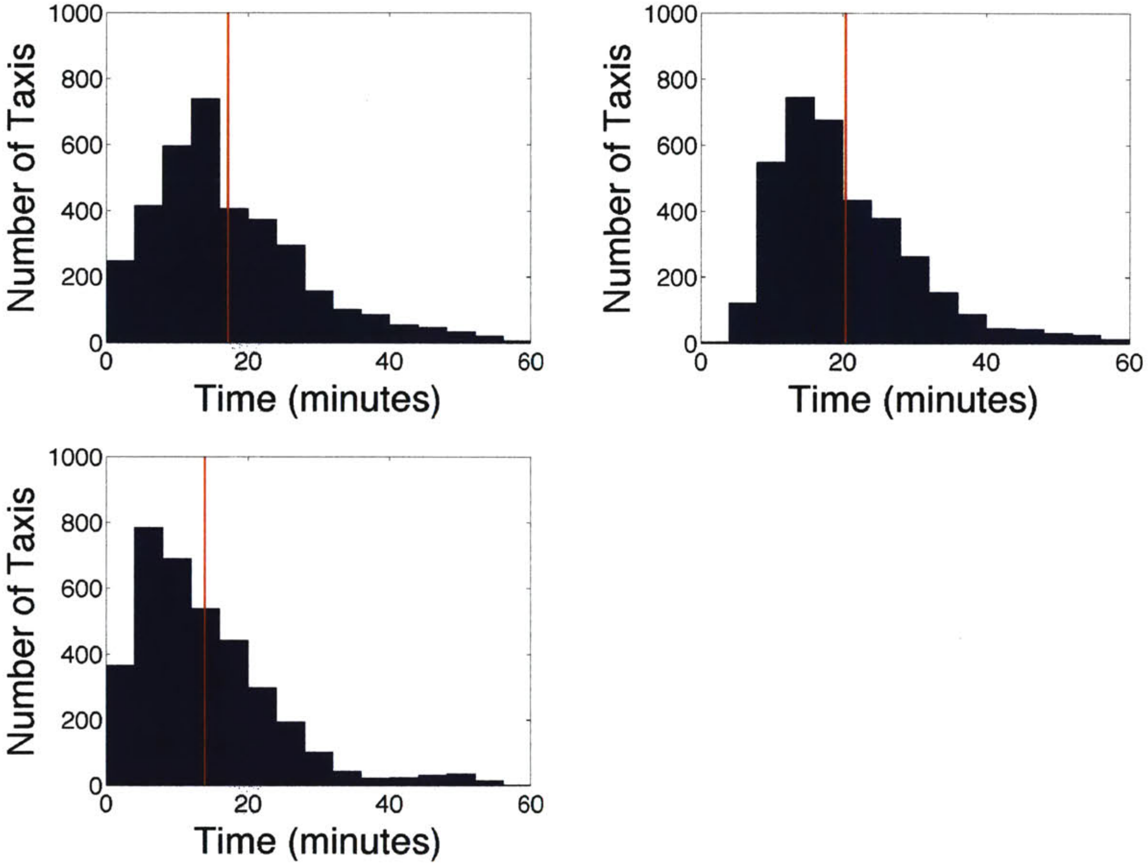
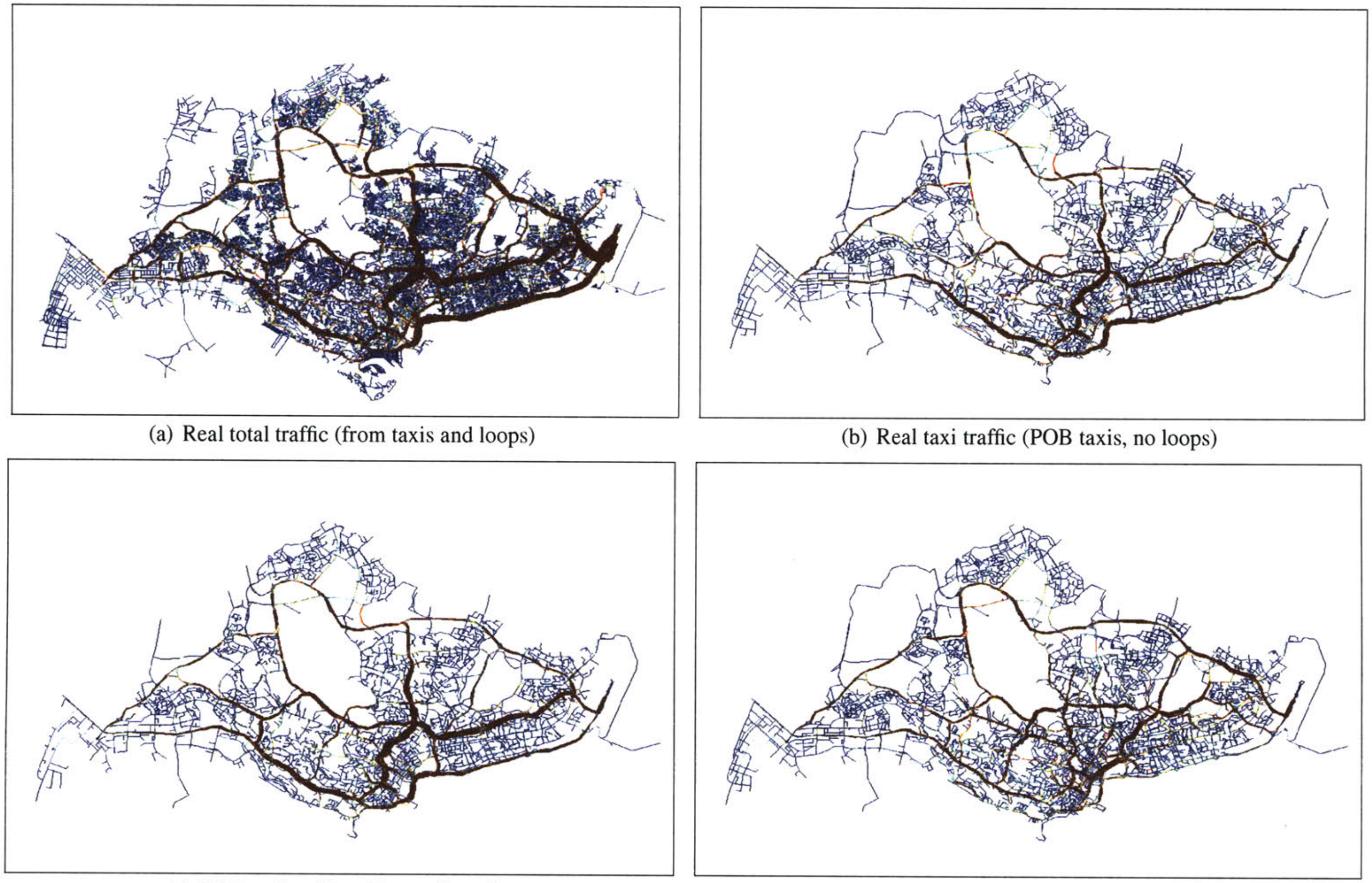


Figure 10-16: Histogram of trip lengths. Real assignment (top left), greedy assignment (top right), and the social optimum assignment (bottom left). We used the total 3,600 POB taxis that start their trips between 08:00∼08:15am on August 2, 2010. The red vertical lines indicate the mean.



(a) Real total traffic (from taxis and loops)

(b) Real taxi traffic (POB taxis, no loops)

(c) POB taxi traffic with greedy paths

(d) POB taxi traffic with socially optimal paths

Figure 10-17: Traffic flow between 8 am ∼ 8:15 am on August 2, 2010. The width of the line indicates the amount of traffic flow during the 15-minute slot. The amount of flow is also color-coded continuously from red (high) to blue (low). Red encodes high traffic flows; blue encodes low traffic flows. The thicker the red lines, the higher the traffic flows. The flow pattern for real taxi traffic (b) is similar to the total traffic (a). We see that the taxis' flow at social optimum is very different from the real taxi traffic, and the social optimum assignment is distributing the taxi traffic across roads (d), whereas the greedy assignment increases the traffic on the popular roads (c).

10.3 Synopsis

In this chapter, we have presented a traffic information and congestion-aware route planning system that supports multiple drivers on a road network. We have shown that taxi probes can be used to infer traffic patterns such as congestion patterns, traffic hotspots, and historical traffic volumes. Using the learned congestion model, we developed a multi-agent route planning system capable of computing socially-optimal paths. We evaluated the performance of this system using traffic data recorded in Singapore by 16,000 taxis. The experiments show that socially-optimal congestion-aware routing achieves 15% travel time reduction. In our experiments, we controlled a fraction of the traffic (the taxi traffic due to POB accounted for approximately 10% of the total traffic; this fraction was estimated by comparing taxi volume counts to loop detector counts.) We hypothesize that the savings would be larger if a larger fraction of the overall traffic can be controlled.

Chapter 11

Conclusions, Lessons Learned, and Future Work

In this thesis, we develop congestion-aware routing algorithms based on data-driven traffic modeling. These highly efficient algorithms make possible much improved large-scale traffic planning. We have developed efficient route planning algorithms that consider (1) uncertainties in travel time and (2) the congestion effect according to the drivers' path choices. The algorithms are designed in such a way that they can be implemented in a large network with a large number of agents. The traffic models for the algorithm are estimated using the real data from inductive loop detectors and a fleet of roving taxis, and they capture important characteristics of traffic conditions, including uncertainty in travel time and congestion created by agents' road usage. We design the predictive model to infer the general traffic flow from taxi data and loop detector data. Using the developed algorithms and data-driven congestion models, we implement a traffic routing system for multiple agents. We evaluate our algorithm using the sensor data. We demonstrate that we can achieve the users' goals while reducing the travel time for the users and for the society when our algorithms are applied. By running very large scale experiments using the real data from 16,000 taxis and 10,000 loop detectors, we show that the city-scale congestion can be mitigated by planning drivers' routes, while incorporating the congestion effects generated by their route choices.

Though the work was motivated by the multi-agent navigation problem in a stochastic network, the algorithms presented in this chapter are general; hence, they can be applied in many areas of motion planning problems, as well as uncertainty or risk management problems in many more areas.

Lessons Learned In the course of this research we have learned several valuable lessons concerning data processing, analysis, and algorithm development.

Firstly, we have learned that the modeling of the traffic and prediction algorithms should be based on data. The data we work on always have anomalies and outliers. In addition, the data do not correspond perfectly to the theoretical expectation. For example, the fundamental diagram theory describes an ideal situation but the data show more complicated relationships. We needed to modify the previous models based on our data. The data indicate what the important factors are as well as what and how we should model with the data. We should look at data and accurately capture what is indicated.

A second lesson we have learned is that a combination of ideas from different disciplines can generate powerful methods. The concept of consensus controller was borrowed from multiple robot coordination and applied to solve a very large-scale problem in a road network.

The third lesson we have learned is that efficient algorithm development is crucial for large-scale multi-agent route planning problems, and that distributed algorithms and implementation make large-scale applications possible. We have seen that the computational cost can be reduced using efficient algorithms that apply the specific computational structure of the problems. We have learned that a decentralized algorithm can find a global solution using local information. The appropriate decomposition of the problem allows the local knowledge to be sufficient for computing globally optimal solutions, and it enables decentralized computation for large-scale traffic route planning. We have learned that the formulation of the problem is general so that it can be used in many settings. We have developed our algorithm in traffic planning settings. However, we realized that many other problems can be modeled in a similar way. Robotic motion planning and financial risk management are such examples.

Future Work This thesis is a step towards city-scale congestion detection and control in urban environments using dynamic sensors. The research in this thesis also points toward several lines of future work.

One interesting direction for future research is to improve the generalization techniques for traffic flow estimation. Our flow estimation algorithm used similarity information between road segments. If we have more information about the road category, etc., we can try to use it to classify the roads. We can also use the new information for finding an optimal weight for similarities. This study has been conducted using Singapore data. We envision running our algorithm in different cities.

Another avenue for future work is to develop a congestion-aware dynamic road pricing scheme as an on-line control alternative for congestion. Theoretically, it is well known that the marginal cost should be charged to make selfish agents use the social optimum paths. However, the values of time delay perceived by different users may be different. Thus, we need to quantify how much the time value corresponds to the money value, and use it to implement the congestion pricing algorithms.

It would also be interesting to identify various extensions of this work, including applications of the dynamic congestion model. In a multi-agent route planning algorithm area, we are working on the multi-agent routing problem considering spatio-temporal interaction between cars. In this thesis, we developed a multi-agent routing algorithm using a predefined number of paths for each agent. We can also work on changing the set of paths in the process of computing the paths. To create a multi-agent route planning algorithm dealing with both uncertainty and congestion would be a good research direction as well.

We studied the mobility pattern in terms of the origin and destination. This information is valuable information for road traffic management. We have ground-truth information for each taxi's origin and destination. Currently, we have shown the effect of our algorithm on city-scale deployment using taxis. Estimation of the origin and destination distribution of other drivers using taxi data is a future research problem. One approach is to use subsample taxi data for the estimation and validate the estimation algorithm using the test set. Accurate estimation of the origin and destination distribution of general traffic will enable more realistic simulation of our multi-agent path planning algorithms for general traffic.

Bibliography

[1] 75 years of the fundamental diagram for traffic flow theory: Greenshields symposium | blurbs | main. http://www.trb.org/Main/Blurbs/165625.aspx.

[2] Google maps api.

[3] Highway performance monitoring system, federal highway administration, `http://www.fhwa.dot.gov/policyinformation/hpms.cfm`.

[4] National average speed database, INRIX, `www.inrix.com`.

[5] Tilera, `http://www.tilera.com`.

[6] Traffic detector handbook: Third edition, fhwa-hrt-06-108, october 2006, `http://www.fhwa.dot.gov/publications/research/operations/its/06108/index.cfm`.

[7] The 1995 national personal transportation survey (NPTS), `http://npts.ornl.gov/npts/1995/Doc/publications.shtml`. 1995.

[8] the new 2000 national household travel survey (NHTS), `http://www.bts.gov/programs/national_household_travel_survey/`. 2000.

[9] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato Fonseca F. Werneck. VC-dimension and shortest path algorithms. In *38th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 690–699, 2011.

[10] Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

[11] Ittai Abraham and Cyril Gavoille. Object location using path separators. In *25th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 188–197, 2006.

[12] E. Altman and H. Kameda. Equilibria for multiclass routing in multi-agent networks. In *Proceedings of the 40th IEEE Conference on Decision and Control, 2001*, volume 1, pages 604–609 vol.1. IEEE, 2001.

[13] Seungkirl Baek, Hyunmyung Kim, and Yongtaek Lim. Multiple-Vehicle Origin–Destination matrix estimation from traffic counts using genetic algorithm. *Journal of Transportation Engineering*, 130(3):339–347, May 2004.

[14] H. Bar-Gera and A. Luzon. Differences among route flow solutions for the user-equilibrium traffic assignment problem. *Journal of transportation engineering*, 133:232, 2007.

[15] R. Barlovic, L. Santen, A. Schadschneider, and M. Schreckenberg. Metastable states in cellular automata for traffic flow. *The European Physical Journal B*, 5(3):793–800, October 1998.

[16] Yair Bartal, Lee-Ad Gottlieb, Tsvi Kopelowitz, Moshe Lewenstein, and Liam Roditty. Fast, precise and dynamic distance queries. In *22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 840–853, 2011.

[17] Holger Bast, Stefan Funke, Peter Sanders, and Dominik Schultes. Fast routing in road networks with transit nodes. *Science*, 316(5824):566, 2007.

[18] Reinhard Bauer and Daniel Delling. SHARC: Fast and robust unidirectional routing. *ACM Journal of Experimental Algorithmics*, 14, 2009.

[19] M.G.H. Bell and C. Cassir. Risk-averse user equilibrium traffic assignment: an application of game theory. *Transportation Research Part B: Methodological*, 36(8):671–681, 2002.

[20] D. P. Bersekas and J. N. Tsitsiklis. Parallel and distributed computation: Numerical methods. *Athena Scientific,*, 1997.

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

[22] M. Bierlaire and F. Crittin. An efficient algorithm for Real-Time estimation and prediction of dynamic OD tables. *Operations Research*, 52(1):116–127, January 2004. ArticleType: research-article / Full publication date: Jan. - Feb., 2004 / Copyright 2004 INFORMS.

[23] P. Borokhov, S. Blandin, S. Samaranayake, O. Goldschmidt, and A. Bayen. An adaptive routing system for location-aware mobile devices on the road network. In *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*, pages 1839–1845, 2011.

[24] Tom Caldwell. On finding minimum routes in a network with turn penalties. *Communications of the ACM*, 4(2), 1961.

[25] A. Cantoni. Optimal curve fitting with piecewise linear functions. *IEEE Transactions on Computers*, C-20(1):59– 67, January 1971.

[26] 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.

[27] Ennio Cascetta and Sang Nguyen. A unified framework for estimating or updating origin/destination matrices from traffic counts. *Transportation Research Part B: Methodological*, 22(6):437–455, December 1988.

[28] R.K. Cheung. Iterative methods for dynamic stochastic shortest path problems. *Naval Research Logistics (NRL)*, 45(8):769–789, 1998.

[29] N. Chiabaut, C. Buisson, and L. Leclercq. Fundamental diagram estimation through passing rate measurements in congestion. *Intelligent Transportation Systems, IEEE Transactions on*, 10(2):355–359, 2009.

[30] F Chudak. Static traffic assignment problem: A comparison between beckmann (1956) and nesterov & de palma (1998) models." conference proceedings of 7th swiss transport research conference, september 12, 2007-September 14, 2007, Monte-Verita, ascona, switzerland. (PDF).

[31] J.R. Correa, A.S. Schulz, and N.E. Stier-Moses. Selfish routing in capacitated networks. *Mathematics of Operations Research*, pages 961–976, 2004.

[32] A.I. Corra, A. Langevin, and L.M. Rousseau. Scheduling and routing of automated guided vehicles: A hybrid approach. *Computers & operations research*, 34(6):1688–1707, 2007.

[33] T. G Crainic, M. Florian, and Y. Noriega. Computing shortest paths with logistic constraints. Technical report, Working paper, 2005.

[34] M. Cremer and M. Papageorgiou. Parameter identification for a traffic flow model. *Automatica*, 17(6):837–843, 1981.

[35] Carlos F. Daganzo and Nikolas Geroliminis. An analytical approximation for the macroscopic fundamental diagram of urban traffic. *Transportation Research Part B: Methodological*, 42(9):771–781, November 2008.

[36] C.F. Daganzo and Y. Sheffi. On stochastic models of traffic assignment. *Transportation Science*, 11(3):253–274, 1977.

[37] X. Dai, M.A. Ferman, and R.P. Roesser. A simulation evaluation of a real-time traffic information system using probe vehicles. In *Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE*, volume 1, pages 475–480, 2003.

[38] Tim Lomax David Schrank and Shawn Turner. TTI's 2010 urban mobility report, texas transportation institute, the texas a&m university system, `http://mobility.tamu.edu`. 2010.

[39] Juan de Dios Ortzar and Luis G. Willumsen. Modelling transport, third edition. *John Wiley & Sons*, 2001.

[40] S. Devarajan. A note of network equilibrium and noncooperative games. *Transportation Research Part B: Methodological*, 15(6):421–426, 1981.

[41] Wolf Dietrich E. Cellular automata for traffic simulations. *Physica A: Statistical Mechanics and its Applications*, 263(14):438–451, February 1999.

[42] AiLing Ding, XiangMo Zhao, and LiCheng Jiao. Traffic flow time series prediction based on statistics learning theory. In *Intelligent Transportation Systems, 2002. Proceedings. The IEEE 5th International Conference on*, pages 727 – 730, 2002.

[43] David Eppstein and Michael T. Goodrich. Studying (non-planar) road networks through an algorithmic lens. In *16th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (GIS)*, page 16, 2008.

[44] S. Even, A. Itai, and A. Shamir. On the complexity of time table and multicommodity flow problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 184–193, 1975.

[45] Y.Y. Fan, R.E. Kalaba, and J.E. Moore II. Shortest paths in stochastic networks with correlated link costs. *Computers & Mathematics with Applications*, 49(9-10):1549–1564, 2005.

[46] N. Farhi, M. Goursat, and J. P Quadrat. Derivation of the fundamental traffic diagram for two circular roads and a crossing using minplus algebra and petri net modeling. In *44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05*, pages 2119- 2124. IEEE, December 2005.

[47] Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, October 2004.

[48] M.A. Ferman, D.E. Blumenfeld, and X. Dai. A simple analytical model of a probe-based traffic information system. In *Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE*, volume 1, pages 263–268, 2003.

[49] T.L. Friesz, D. Bernstein, T.E. Smith, R.L. Tobin, and BW Wie. A variational inequality formulation of the dynamic network user equilibrium problem. *Operations Research*, 41(1):179–191, 1993.

[50] C. Gawron. An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model. *International Journal of Modern Physics C*, 9:393–408, 1998.

[51] Robert Geisberger, Moritz Kobitzsch, and Peter Sanders. Route planning with flexible objective functions. In *12th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 124–137, 2010.

[52] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *7th International Workshop on Experimental Algorithms (WEA)*, pages 319–333, 2008.

[53] Robert Geisberger and Christian Vetter. Efficient routing in road networks with turn costs. In *10th International Symposium on Experimental Algorithms (SEA)*, pages 100–111, 2011.

[54] Nikolai Golovchenko. Least-squares fit of a continuous piecewise linear function, August 2004.

[55] M.C. González, C.A. Hidalgo, and A.L. Barabási. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, 2008.

[56] J.S. Greenfeld. Matching GPS observations to locations on a digital map. In *81th Annual Meeting of the Transportation Research Board*, volume 1, 2002.

[57] Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel H. M. Smid. Approximate distance oracles for geometric spanners. *ACM Transactions on Algorithms*, 4(1), 2008.

[58] Martin L. Hazelton. Estimation of origin-destination matrices from link flows on uncongested networks. *Transportation Research Part B: Methodological*, 34(7):549–566, September 2000.

[59] Martin L. Hazelton. Inference for origin-destination matrices: estimation, prediction and reconstruction. *Transportation Research Part B: Methodological*, 35(7):667–676, August 2001.

[60] Martin L. Hazelton. Statistical inference for time varying origin-destination matrices. *Transportation Research Part B: Methodological*, 42(6):542–552, July 2008.

[61] Martin L. Hazelton. Statistical inference for transit system Origin-Destination matrices. *Technometrics*, 52(2):221–230, May 2010.

[62] J.C. Herrera and A.M. Bayen. Traffic flow reconstruction using mobile sensors and loop detector data. In *87th TRB Annual Meeting*, 2007.

[63] A. Hofleitner and A. Bayen. Optimal decomposition of travel times measured by probe vehicles using a statistical traffic flow model. In *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*, pages 815–821, 2011.

[64] E. Horvitz, J. Apacible, R. Sarin, and L. Liao. Prediction, expectation, and surprise: Methods, designs, and study of a deployed traffic forecasting service. In *Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 244–257, 2005.

[65] 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.

[66] Bret Warren Hull. Opportunistic sensing and mobile data delivery in the cartel system. *MIT Ph. D. Thesis*, 2010.

[67] Texas Transportation Institute. Annual urban mobility report. 2010.

[68] Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, *Column Generation*, pages 33–65. Springer-Verlag, New York, 2005.

[69] O. Jahn, R. H Möhring, A. S Schulz, and N. E Stier-Moses. System-optimal routing of traffic flows with user constraints in networks with congestion. *Operations Research*, 53(4):600–616, 2005.

[70] Ken-ichi Kawarabayashi, Philip N. Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *38th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 135–146, 2011.

[71] F. Kelly. Charging and rate control for elastic traffic. *European transactions on Telecommunications*, 8(1):33–37, 1997.

[72] F.P. Kelly, A.K. Maulloo, and D.K.H. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 49(3):237–252, 1998.

[73] A. Kotsialos, M. Papageorgiou, C. Diakaki, Y. Pavlis, and F. Middelham. Traffic flow modeling of large-scale motorway networks using the macroscopic modeling tool METANET. *Intelligent Transportation Systems, IEEE Transactions on*, 3(4):282 – 292, December 2002.

[74] S. Krauss, P. Wagner, and C. Gawron. Metastable states in a microscopic model of traffic flow. *Physical Review E*, 55(5):5597–5602, May 1997.

[75] W.H.K. Lam, Z.Y. Gao, K.S. Chan, and H. Yang. A stochastic user equilibrium assignment model for congested transit networks. *Transportation Research Part B: Methodological*, 33(5):351–368, June 1999.

[76] Jorge A. Laval and Ludovic Leclercq. A mechanism to describe the formation and propagation of stop-and-go waves in congested freeway traffic. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1928):4519 –4541, October 2010.

[77] Larry J. LeBlanc and Mustafa Abdulaal. A comparison of user-optimum versus system-optimum traffic assignment in transportation network design. *Transportation Research Part B: Methodological*, 18(2):115–121, April 1984.

[78] Jung Hoon Lee, Beom Hee Lee, and Myoung Hwan Choi. A real-time traffic control scheme of multiple AGV systems for collision free minimum time motion: a routing table approach. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 28(3):347–358, May 1998.

[79] Thomas C. M. Lee. An introduction to coding theory and the two–part minimum description length principle. *International Statistical Review*, 69:169–183, 2001.

[80] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:422–431, 1988.

[81] T. Li. Modelling traffic flow with a time-dependent fundamental diagram. *Mathematical methods in the applied sciences*, 27(5):583–601, 2004.

[82] Sejoon Lim, Hari Balakrishnan, David Gifford, Samuel Madden, and Daniela Rus. Stochastic motion planning and applications to traffic. In *Proceedings of the Eighth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Guanajuato, Mexico, December 2008.

[83] Sejoon Lim, Hari Balakrishnan, David Gifford, Samuel Madden, and Daniela Rus. Stochastic motion planning and applications to traffic. *The International Journal of Robotics Research*, 30(6):699 –712, May 2011.

[84] Sejoon Lim and Daniela Rus. Stochastic distributed multi-agent planning and applications to traffic. *International Conference on Robotics and Automation (ICRA), May 14-18, St. Paul, Minnesota*, 2012.

[85] Sejoon Lim and Daniela Rus. Stochastic motion planning with path constraints and application to optimal agent, resource and route planning. *International Conference on Robotics and Automation (ICRA), May 14-18, St. Paul, Minnesota*, 2012.

[86] Sejoon Lim, Christian Sommer, Evdokia Nikolova, and Daniela Rus. Practical route planning under delay uncertainty: Stochastic shortest path queries. *Robotics: Science and Systems Conference*, 2012.

[87] T. Litman. Measuring transportation: traffic, mobility and accessibility. 2003.

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

[89] S.H. Low and D.E. Lapsley. Optimization flow controll: basic algorithm and convergence. *IEEE/ACM Transactions on Networking (TON)*, 7(6):861–874, 1999.

[90] Sven Maerivoet and Bart De Moor. Transportation planning and traffic flow models. *arXiv:physics/0507127*, July 2005.

[91] J. Markoff. Google cars drive themselves, in traffic. *The New York Times*, 10:A1, 2010.

[92] M.M. Minderhoud, H. Botma, and P.H.L. Bovy. Assessment of roadway capacity estimation methods. *Transportation Research Record: Journal of the Transportation Research Board*, 1572(-1):59–67, 1997.

[93] Andrew Moore. K-means and hierarchical clustering. `http://www.autonlab.org/tutorials/kmens11.pdf`, Nov 2001. Accessed Mar 30, 2011.

[94] Laurent Flindt Muller and Martin Zachariasen. Fast and compact oracles for approximate distances in planar graphs. In *15th European Symposium on Algorithms (ESA)*, pages 657–668, 2007.

[95] I. Murthy and S. Sarkar. A relaxation-based pruning technique for a class of stochastic shortest path problems. *Transportation Science*, 30(3):220–236, 1996.

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

[97] I. Murthy and S. Sarkar. Stochastic shortest path problems with piecewise-linear concave utility functions. *Management Science*, pages 125–136, 1998.

[98] Kai Nagel, Peter Wagner, and Richard Woesler. Still flowing: Approaches to traffic flow and traffic jam modeling. *Operations Research*, 51(5):681–710, 2003. ArticleType: research-article / Full publication date: Sep. - Oct., 2003 / Copyright 2003 INFORMS.

[99] Paul Newson and John Krumm. Hidden markov map matching through noise and sparseness. *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 336–343, 2009.

[100] Evdokia Nikolova. Approximation algorithms for reliable stochastic combinatorial optimization. In *13th International Workshop on Approximation, Randomization, and Combinatorial Optimization (APPROX)*, pages 338–351, 2010.

[101] Evdokia Nikolova, Matthew Brand, and David R. Karger. Optimal route planning under uncertainty. In *16th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 131–141, 2006.

[102] Evdokia Nikolova, Jonathan A. Kelner, Matthew Brand, and Michael Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *14th European Symposium on Algorithms (ESA)*, pages 552–563, 2006.

[103] Bureau of Public Road. Traffic assignment manual. 1964.

[104] Urban Redevelopment Authority of Singapore. List of postal districts. http://www.ura.gov.sg/realEstateWeb/resources/misc/list_of_postal_districts.htm. Accessed Mar 30, 2011.

[105] Ariel Orda, Raphael Rom, and Nahum Shimkin. Competitive routing in multiuser communication networks. *IEEE/ACM Trans. Netw.*, 1(5):510–521, October 1993.

[106] M. Papageorgiou, M. Ben-Akiva, J. Bottom, P.H.L. Bovy, SP Hoogendoorn, N.B. Hounsell, A. Kotsialos, and M. McDonald. ITS and traffic management. *Handbooks in Operations Research and Management Science*, 14:715–774, 2007.

[107] P. Pattanamekar, D. Park, L.R. Rilett, J. Lee, and C. Lee. Dynamic and stochastic shortest path in transportation networks with two components of travel time uncertainty. *Transportation Research Part C: Emerging Technologies*, 11(5):331–354, 2003.

[108] T. Pavlidis and S. L Horowitz. Segmentation of plane curves. *IEEE Transactions on Computers*, C-23(8):860– 870, August 1974.

[109] G.H. Polychronopoulos and J.N. Tsitsiklis. Stochastic shortest path problems with recourse. *Networks*, 27(2):133–143, 1996.

[110] W.B. Powell and Y. Sheffi. The convergence of equilibrium algorithms with predetermined step sizes. *Transportation Science*, 16(1):45, 1982.

[111] C. Quek, M. Pasquier, and B.B.S. Lim. POP-TRAFFIC: a novel fuzzy neural approach to road traffic analysis and prediction. *Intelligent Transportation Systems, IEEE Transactions on*, 7(2):133 - 146, June 2006.

[112] Adam Recchia and James C. Hadfield. Regional truck route study. *Southeastern Regional Planning and Economic Development District*, 2009.

[113] E.S. Richardson A.J. Ampt and A. Meyburg. Survey methods for transport planning. *Eucalyptus Press*, 1995.

[114] T. Roughgarden and . Tardos. How bad is selfish routing? *Journal of the ACM (JACM)*, 49(2):236–259, 2002.

[115] Peter Sanders and Dominik Schultes. Highway hierarchies hasten exact shortest path queries. In *13th European Symposium on Algorithms (ESA)*, pages 568–579, 2005.

[116] Ralf-Peter Schäfer. IQ routes and HD traffic: technology insights about TomTom's time-dynamic navigation concept. In *7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 171–172, 2009.

[117] M. Schwager, D. Rus, and J.J. Slotine. Decentralized, adaptive coverage control for networked robots. *The International Journal of Robotics Research*, 28(3):357–375, 2009.

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

[119] H. Slavin, J. Brandon, and A. Rabinowicz. An empirical comparison of alternative user equilibrium traffic assignment methods. In *Proceedings of the European Transport Conference*, 2006.

[120] Christian Sommer, Elad Verbin, and Wei Yu. Distance oracles for sparse graphs. In *50th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 703–712, 2009.

[121] A. Stathopoulos and M.G. Karlaftis. A multivariate state space approach for urban traffic flow modeling and prediction. *Transportation Research Part C: Emerging Technologies*, 11(2):121–135, 2003.

[122] VJ Stephanedes, P.G. Michalopoulos, and R.A. Plum. Improved estimation of traffic flow for Real-Time control (Discussion and closure). *Transportation Research Record*, (795), 1981.

[123] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, 2004.

[124] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005.

[125] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *The 2005 DARPA Grand Challenge*, pages 1–43, 2007.

[126] TomTom. How TomTom's HD Traffic and IQ Routes data provides the very best routing — travel time measurements using GSM and GPS probe data. White Paper online.

[127] J. Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers, Part II*, 1(36):352–362, 1952.

[128] Michael P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[129] 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.

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

[131] David Wilkie, Ming Lin, and Dinesh Manocha. Self-Aware traffic route planning. 2011.

[132] Stephan Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):363–380, 2002.

[133] D. B Work, S. Blandin, O. P Tossavainen, B. Piccoli, and A. M Bayen. A traffic model for velocity data assimilation. *Applied Mathematics Research eXpress*, 2010(1):1, 2010.

[134] D.B. Work, O.P. Tossavainen, S. Blandin, A.M. Bayen, T. Iwuchukwu, and K. Tracton. An ensemble kalman filtering approach to highway traffic estimation using GPS enabled mobile devices. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 5062–5068, 2008.

[135] Chun-Hsin Wu, Jan-Ming Ho, and D.T. Lee. Travel-time prediction with support vector regression. *Intelligent Transportation Systems, IEEE Transactions on*, 5(4):276 - 281, December 2004.

[136] Ning Wu. A new approach for modeling of fundamental diagrams. *Transportation Research Part A: Policy and Practice*, 36(10):867–884, December 2002.

[137] W. Wu and Q. Ruan. A hierarchical approach for the shortest path problem with obligatory intermediate nodes. In *Signal Processing, 2006 8th International Conference on*, volume 4.

[138] J. Yoon, B. Noble, and M. Liu. Surface street traffic estimation. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 220–232, 2007.

[139] H.M. Zhang and T. Kim. A car-following theory for multiphase vehicular traffic flow. *Transportation Research Part B: Methodological*, 39(5):385–399, 2005.

[140] L. Zhi-Peng, G. Xiao-Bo, and L. Yun-Cai. An improved car-following model for multiphase vehicular traffic flow and numerical tests. *Communications in Theoretical Physics*, 46:367, 2006.