The International Journal of
Robotics Research
30(6) 699–712
© The Author(s) 2011
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0278364910386259
ijr.sagepub.com

Stochastic motion planning and applications to traffic

Sejoon Lim, Hari Balakrishnan, David Gifford, Samuel Madden and Daniela Rus

Abstract

This paper presents a stochastic motion planning algorithm and its application to traffic navigation. The algorithm copes with the uncertainty of road traffic conditions by stochastic modeling of travel delay on road networks. The algorithm determines paths between two points that optimize a cost function of the delay data probability distribution. It can be used to find paths that maximize the probability of reaching a destination within a particular travel deadline. For such problems, standard shortest-path algorithms do not work because the optimal substructure property does not hold. We evaluate our algorithm using both simulations and real-world drives, using delay data gathered from a set of taxis equipped with global positioning system sensors and a wireless network. Our algorithm can be integrated into on-board navigation systems as well as route-finding websites, providing drivers with good paths that meet their desired goals.

Keywords
Motion planning, Stochastic shortest path, Traffic navigation, Algorithm

1. Introduction

This paper presents and evaluates an algorithm for planning the motion of vehicles (autonomous or human-driven) on roadways in the face of traffic delays. Rather than model road delays statically, as in current on-board navigation systems and web-based mapping services, our algorithm uses past observations of actual delays on road segments to model these delays as probability distributions. The algorithm minimizes a user-specified cost function of the delay distribution. We investigate three cost functions in detail, particularly one that is equivalent to maximizing the likelihood of reaching a destination within a specified travel deadline (Lim et al. 2008).

Our work provides a planning system that can be used by robots, human drivers, or other agents that require congestion-aware planning. The system is a useful addition to on-board navigation systems using computer-aided automation to provide good paths that meet desired travel goals (e.g., 'when should you leave, and what path should you take, to reach the airport by 8am with high probability?'); it can also operate with web-based mapping services. We view the incorporation of traffic-aware path computation as an important practical addition in the rapid trend toward computer-assisted driving and autonomous decision-making in vehicles.

Traffic congestion is clearly a serious problem: a recent survey (Schrank and Lomax 2007) estimates that the annual nationwide cost of traffic congestion is US$78 billion, including 4.2 billion hours in lost time and 2.9 billion gallons in wasted fuel. Drivers today have little knowledge of historic and real-time traffic congestion on the paths they drive, and even when they do (e.g., from 'live' traffic updates), they generally do not know how to use that information to find good paths. As a result, they often tend to drive sub-optimal routes and often leave well in advance when they need to make an important deadline.

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

The main challenge in planning paths taking traffic delay into account is that these delays are not fixed. The delay

Computer Science and Artificial Intelligence Laboratory, MIT, USA

Corresponding author:
Sejoon Lim
32 Vassar Street Room 32-376, Cambridge, MA 02139, USA
Email: sjlim@csail.mit.edu

on a road segment is best modeled as a probability distribution; in addition, this distribution typically depends on a number of factors, such as time-of-day, whether it is a working day or not, events such as concerts or sporting events, weather, etc. The shortest-distance path is often not the best path to use if one seeks to minimize the expected travel time or maximize the probability of reaching the destination by a certain time. Our algorithm uses historic observations of travel delays on road segments at different times of day to produce delay distributions (indexed by time-of-day). We posit that this information, together with real-time updates of extraneous conditions (such as accidents), is invaluable (and sufficient) to compute good paths that meet user-specified goals. Given the probability distributions of delays on segments, finding good paths requires more than a shortest-path computation, because the 'optimal substructure' property does not hold as explained in Nikolova et al. (2006a) (i.e. if the best path from S to T goes through X, it does not follow that the sub-path of this path from S to X is itself the best S–X path).

We have implemented our algorithm and evaluated it by first modeling the historic delays using data from the CarTel vehicular testbed (Hull et al. 2006), a network of 28 taxis. The data consists of travel times organized by road segment and by time-of-day, yielding statistical profiles for all the road segments. We model the road network as a weighted graph where the nodes represent intersections and the edges represent road segments. An aggregation algorithm combines the road segments into groups to coalesce the important delay characteristics without losing information about alternate paths. Our algorithm has the flavor of searching and pruning the delay statistics on the road network data structure. We evaluate the algorithm and its assumptions using simulation and actual test driving.

1.1. Related Work

Our work is closely related to stochastic planning and stochastic shortest-path algorithms. In Chabini (1998), an efficient algorithm for a dynamic shortest path with time-dependent deterministic edge cost is given. Prior work has considered stochastic shortest paths under the assumption that the travel time follows a known probability distribution (Sigal et al. 1980; Bertsekas and Tsitsiklis 1991; Loui 1983; Wellman et al. 1995; Murthy and Sarkar 1997; Nikolova et al. 2006a,b) 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. Nikolova et al.(2006b) 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.

There have been several approaches to acquiring traffic data. The most prevalent one uses inductor loops installed beneath roads (Xu and Dailey 1995; Chrobok et al. 2001). This is adequate for counting the number of cars that pass a specific location, but it is not suitable for measuring travel time and measurements are possible only on instrumented roads. Recently, global positioning system (GPS) sensors installed in probe vehicles have been used (Sanwal and Walrand 1995; Hull et al. 2006; Yoonet al. 2007). The travel time of vehicles can be measured and recorded for each route segment. In Hull et al. (2006), the CarTel Network nodes that include GPS and wireless communication were described. This system was used to study routing and data delivery from cars.

1.2. Outline

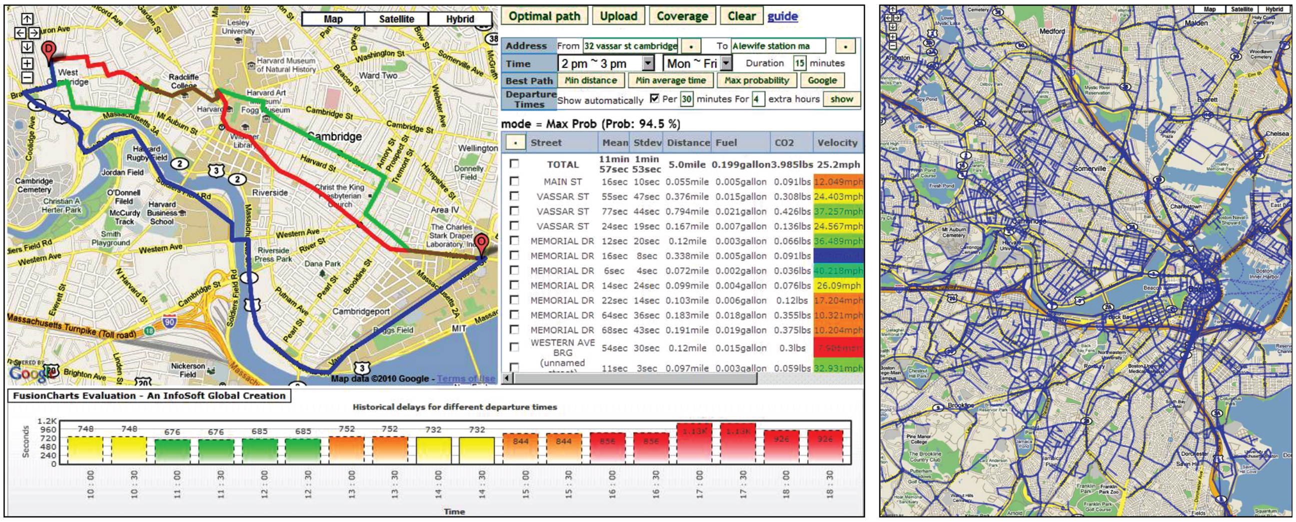
This paper is organized as follows. Section 2 gives an overview of a route planning system for traffic. Section 3 gives the stochastic motion planning formulation, and Section 4 presents our algorithm and gives its correctness and performance bounds. Section 5 evaluates the algorithm via a simulation and also using physical data from the deployment of GPS sensor nodes in taxis.

2. Transportation system context

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

1. a data gathering system included in cars that move in traffic frequently;
2. a data analysis system to compile a historical database of traffic conditions;
3. an algorithm for route planning that uses both historical data and current information;
4. a traffic information system implementation with an appropriate interface.

In Hull et al. (2006) a system called CarTel was developed that uses GPS and wireless communication to collect position and time data from cars. CarTel nodes deployed in 28 taxis since January 2007 collected travel data for the greater Boston area. We have collected over 200 million data points for 600 thousand road segments. This data was organized per road segment to create a historical database of traffic delays.



Address From 32 vassar st cambridge To Alewife station ma

Time 2 pm ~ 3 pm Mon ~ Fri Duration 15 minutes

Departure Times Show automatically ☑ Per 30 minutes For 4 extra hours

mode = Max Prob (Prob: 94.5 %)

Street	Mean	Stdev	Distance	Fuel	CO2	Velocity
TOTAL	11min 57sec	1min 53sec	5.0mile	0.199gallon	3.985lbs	25.2mph
MAIN ST	16sec	10sec	0.055mile	0.005gallon	0.091lbs	12.049mph
VASSAR ST	55sec	47sec	0.376mile	0.015gallon	0.308lbs	24.403mph
VASSAR ST	77sec	44sec	0.794mile	0.021gallon	0.426lbs	37.257mph
VASSAR ST	24sec	19sec	0.167mile	0.007gallon	0.136lbs	24.567mph
MEMORIAL DR	12sec	20sec	0.12mile	0.003gallon	0.066lbs	36.489mph
MEMORIAL DR	16sec	8sec	0.338mile	0.005gallon	0.091lbs	[illegible]
MEMORIAL DR	6sec	4sec	0.072mile	0.002gallon	0.036lbs	40.218mph
MEMORIAL DR	14sec	24sec	0.099mile	0.004gallon	0.076lbs	26.09mph
MEMORIAL DR	22sec	14sec	0.103mile	0.006gallon	0.12lbs	17.204mph
MEMORIAL DR	64sec	36sec	0.183mile	0.018gallon	0.355lbs	10.321mph
MEMORIAL DR	68sec	43sec	0.191mile	0.019gallon	0.375lbs	10.204mph
WESTERN AVE BRG	54sec	30sec	0.12mile	0.015gallon	0.3lbs	7.905mph
(unnamed	11sec	3sec	0.097mile	0.003gallon	0.059lbs	32.931mph

Fig. 1. The user interface to the traffic system with highlighted paths and travel times found by the algorithm described in this paper. Roads with traffic travel time data in our database are also shown to the right.

A stochastic route planning algorithm was developed and implemented. The best route depends on the drivers' goals and is a combination of speed and reliability. A web-based interface (Figure 1) allows users to query the system for traffic conditions and for optimal paths given historical data. The rest of this paper describes the motion planning component 3 of this intelligent traffic system.

3. Problem formulation

3.1. *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 road segment (Figure 1). 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;
2. the delay of each edge is independent of every other edge.

In Section 5, 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:

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

3.2. *Cost functions*

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

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 probability density function (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

$$EC_\pi(d) = \int_{-\infty}^{\infty} u(t-d) f_\pi(t) dt = \int_d^\infty f_\pi(t) dt$$
$$= \{1 - \Phi\left(\frac{d - m_\pi}{\sqrt{v_\pi}}\right)\},$$

where $\Phi(\cdot)$ is the cumulative distribution function (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

$$\varphi_d(\pi) = \frac{d - m_\pi}{\sqrt{v_\pi}}. \tag{1}$$

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 algorithm or the 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, (1), is nonlinear. Our goal for the rest of this paper is to develop an efficient algorithm for finding the maximum probability path given a deadline.

4. Stochastic path planning by parametric optimization

In Nikolova et al. (2006b), 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 (Nikolova et al. 2006b).

4.1. *Transforming the cost function into parametric form*

Let path π be denoted by a point (m_π, v_π) in a rectangular coordinate system, called the *m*–*v* plane, where the horizontal axis represents the mean and the vertical axis represents the variance. The objective of the optimization problem, (1), can be rewritten to show the relation between m_π and v_π as

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

which is a parabola in the *m*–*v* plane with apex at d, where $\varphi_d(\pi)$ is determined by the curvature of the parabola. Thus, the optimal path is the path that lies on the parabola of the smallest curvature. 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 2 (Left)). This suggests finding the optimal path using linear optimization with various combinations of cost coefficients.[1]

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 Nikolova et al. (2006b).

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

Proof 1. Let point P on the *m*–*v* plane represent the optimal path. Then, there is no path point that has a 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 the λ-optimal solution for a given λ. Since the λ-optimal cost satisfies the optimal substructure property, any deterministic shortest-path algorithm (e.g., Dijkstra's algorithm or the A^* search algorithm) will find λ-optimal paths.

4.2. *Exhaustive enumeration*

In Nikolova et al. (2006b), a method for stochastic motion planning was proposed that exhaustively enumerates all of 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 2 (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.

4.3. *Examining probe points*

If we know that a certain search region's best possible outcome is worse than the current optimal solution, we do not

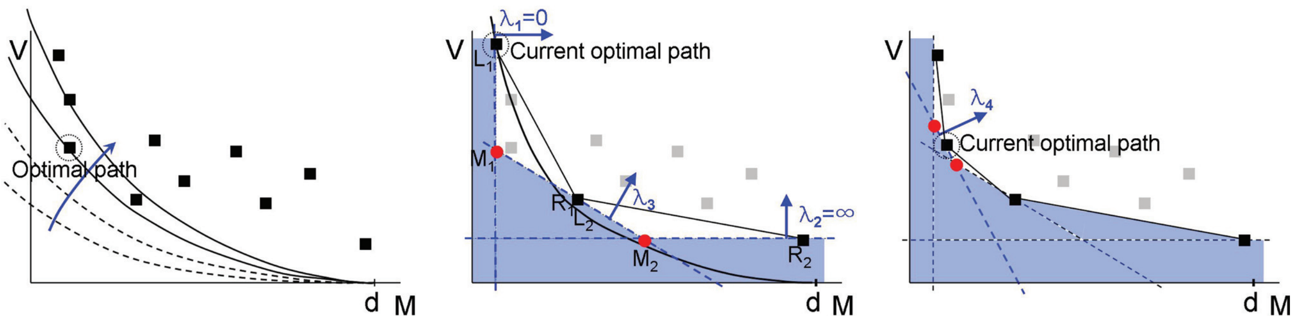


Fig. 2. (Left) Graphical interpretation of the optimal path in the m–v plane. Each square represents a path from the origin to the destination. Equi-probability paths lie on a parabola with an apex at $(d, 0)$ and a curvature of $\frac{1}{\varphi_d(\pi)^2}$. The optimal path is the first point that meets with a parabola as we increase the curvature. (Middle) 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. (Right) 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.

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 2 (Middle).*

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

Proof 2. 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 to the search 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 2 (Right) illustrates a case where the right candidate region $\triangle L_2M_2R_2$ was removed without any execution of the 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 $\triangle L_1M_1R_1$ 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.

4.4. Restricting λ by upper and lower bounds

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

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

Proof 3. 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 the λ-optimal path, $m_{\lambda_u} + \lambda_u v_{\lambda_u} < m_\lambda + \lambda_u v_\lambda$ and $m_{\lambda_u} + \lambda v_{\lambda_u} > m_\lambda + \lambda v_\lambda$. Rewriting these, we get

$$\lambda_u(v_{\lambda_u} - v_\lambda) < m_\lambda - m_{\lambda_u},$$
$$\lambda(v_{\lambda_u} - v_\lambda) > m_\lambda - m_{\lambda_u}. \qquad (3)$$

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 derivative of (2) and its negative inverse

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

From this and (3), and since $d - m_0 > d - m_\lambda$ and $v_\lambda > v_\infty$,

$$\frac{m_\lambda - m_{\lambda_u}}{v_{\lambda_u} - v_\lambda} > \frac{d - m_0}{2v_\infty} > \frac{1}{2}\frac{d - m_\lambda}{v_\lambda}. \qquad (4)$$

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 (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 the λ-optimal solution is the same as the λ_u-optimal solution or not, we can find the optimal solution by searching with $\lambda \le \lambda_u$. □

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 the* 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 the 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

$$\lambda_l(v_{\lambda_l} - v_\lambda) < m_\lambda - m_{\lambda_l}, \tag{5}$$

$$\lambda(v_{\lambda_l} - v_\lambda) > m_\lambda - m_{\lambda_l}. \tag{6}$$

Inequalities (5) and (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 (2) and its negative inverse

$$\lambda_l = -1/\frac{\partial v}{\partial m}\Big|_{m=m_0} = \frac{(d - m_{opt})^2}{2v_{opt}(d - m_0)} \tag{7}$$

From (5) and (7),

$$\frac{m_\lambda - m_{\lambda_l}}{v_{\lambda_l} - v_\lambda} < \frac{\varphi_d(\pi_{opt})^2}{2(d - m_0)}. \tag{8}$$

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

$$\frac{\varphi_d(\pi_{opt})^2}{2(d - m_0)} \le \frac{\varphi_d(\pi_\lambda)^2}{2(d - m_0)} < \frac{\varphi_d(\pi_\lambda)^2}{2(d - m_\lambda)} = \frac{d - m_\lambda}{2v_\lambda} \tag{9}$$

since $d - m_0 > d - m_\lambda$. From (8) and (9),

$$\frac{m_\lambda - m_{\lambda_l}}{v_{\lambda_l} - v_\lambda} < \frac{1}{2}\frac{d - m_\lambda}{v_\lambda}. \tag{10}$$

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

$$\frac{1}{\frac{m_\lambda - m_{\lambda_l}}{d - m_\lambda} + 2} > \frac{1}{2}. \tag{11}$$

From (10) and (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 the 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 1) 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 the 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 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.

4.5. Correctness

Algorithm 1 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 4.2, the algorithm finds the optimal solution in a finite number of searches.

4.6. Running time

As shown by Nikolova et al. (2006b) based on Carstensen (1983), there are a total of $N^{\Theta(\log N)}$ extreme points in the m–v 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

Algorithm 1 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 $bestPath$
```

yielding an average running time of $O(N^2 \log^4 N)$, 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 m–v 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 ith iteration and let $p_0(\,i)$ be the probability that neither candidate region is searched at the ith 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 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, $p_2(\,i)$ becomes smaller than $p_0(\,i)$ at around the $\log N_e$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 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 $\log^2 N$ 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 $\log^2 N$ iterations. This results in a fast increase in $p_0(\,i)$ since its

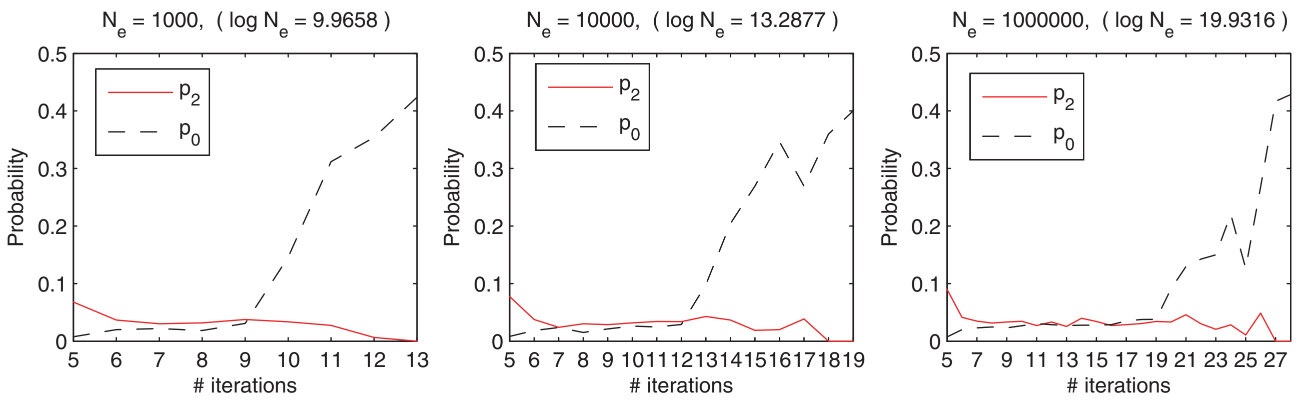


Fig. 3. p_2 and p_0 were calculated from 5000 random convex hulls with N_e extreme points.

increasing rate is related to the decreasing rate of $1 - p_2(i) - p_0(i)$ *by*

$$\frac{\Delta p_0(i)}{\Delta i} = -\frac{\Delta(1 - p_2(i) - p_0(i))}{\Delta i} + \left(-\frac{\Delta p_2(i)}{\Delta i}\right).$$

Thus, $p_0(i)$ *becomes larger than* $p_2(i)$ *around* $i = \log^2 N$. *Figure 3 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. $p_2(i) \le p_0(i), \forall i > n, n = O(\log^2 N)$.

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

Proof 5. 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 $num(\mathcal{N})$ be defined as the total number of candidate regions to search for a tree with a root node $\mathcal{N}$ and let $\mathcal{L}$ and $\mathcal{R}$ be the left and right children nodes of $\mathcal{N}$. Then, $h = O(\log(N^{\log N})) = O(\log^2 N)$ and

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

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

$$E[num(\mathcal{N}) \mid num(\mathcal{L}), num(\mathcal{R})] = 1 + \frac{1 + p_2(i) - p_0(i)}{2}(num(\mathcal{L}) + num(\mathcal{R})).$$

Then, the expectation of $num(\mathcal{N})$ can be expressed as follows by taking the expectation over $num(\mathcal{L})$ and $num(\mathcal{R})$ on the above conditional expectation:

$$E[num(\mathcal{N})] = E[E[num(\mathcal{N}) \mid num(\mathcal{L}), num(\mathcal{R})]] = 1 + \frac{1 + p_2(i) - p_0(i)}{2}(E[num(\mathcal{L})] + E[num(\mathcal{R})]). \quad (12)$$

Since the expectation of $num(\mathcal{N})$ 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 $\mathcal{N}$ whose height is h and whose two children are $\mathcal{L}$ and $\mathcal{R}$, we have $E[num(\mathcal{N})] = num_e(h)$ and $E[num(\mathcal{L})] = E[num(\mathcal{R})] = num_e(h-1)$. From (12), $num_e(h)$ can be written as

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

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

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

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 1 is* $O(N^2 \log^4 N)$.

Proof 6. The total running time is determined by considering the computation before and after $p_2(i) \le p_0(i)$ is satisfied separately. By Lemma 1 the computational cost before $p_2(i) \le p_0(i)$ 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 $p_2(i) \le p_0(i)$, the computational cost after $p_2(i) \le p_0(i)$ is satisfied is $O(N^2 \log^4 N)$ by Lemma 2. Thus, the overall running time is $O(N^2 \log^4 N)$. □

4.7. 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 a 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 1 should be changed to

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

and lines 23 and 25 of Algorithm 1 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.

4.8. 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:

$$v_\pi = \frac{1}{\varphi^2}\{(\,desired\ arrival\ time - departure\ time) - m_\pi\}^2, \tag{13}$$

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

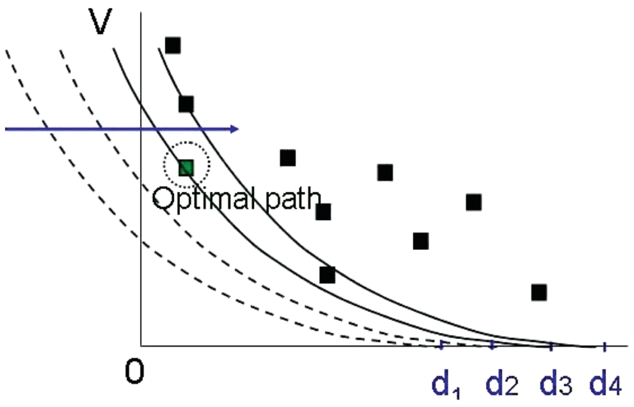


Fig. 4. For a given probability of making a deadline the curvature of the parabola is fixed. There is no path having the deadline as small as d_1 or d_2. The optimal path is indicated as green which has the smallest deadline among all the pathes, d_3. It has the latest departure time since *departure time = desired arrival time – 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 (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 the *m–v* plane. This can be proven with similar arguments with those in Proof 4.1. Thus, we can apply the probe point method we developed in Section 4 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 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). 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 trade off between higher probability of making the deadline and later departure time. Note that all of 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 4.7 to find a sub-optimal solution within a tolerable departure time range.

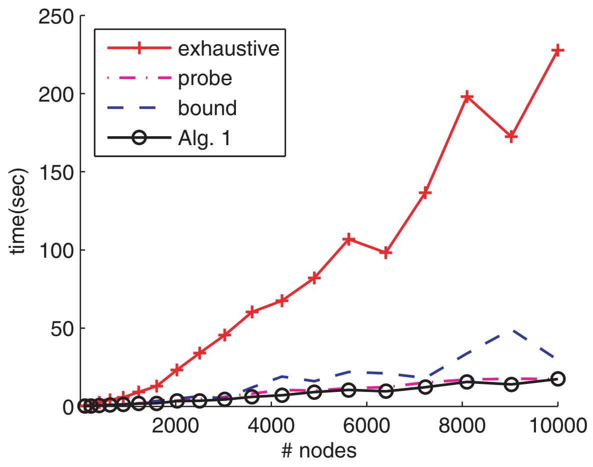


Fig. 5. Running time measured at the square bidirectional grid structure where each edge has a random mean and variance between 0 and 1 with a deadline of half grid size. The curve 'exhaustive' is the exhaustive λ-optimal search, 'probe' is just applying the candidate-region probing method, 'bound' is just applying the bounds of λ and 'Alg. 1' is Algorithm 1.

5. Algorithm evaluation

We have evaluated empirically the performance of Algorithm 1 against the exhaustive λ-optimal search proposed in Nikolova et al. (2006b) using simulation data and real data from the taxi database.

5.1. Experimental data

Grid structure. The simulation data set is a square bidirectional grid structure where each edge has a random mean and variance uniformly distributed between 0 and 1. Grid structures of size from 10×10 to 100×100 are used. The origin and destination are two diagonally opposite points.
Physical road network. The Delay Statistics Map built using the taxi database is used as a physical test bed. The map has about 29,000 nodes and 39,000 edges. It is dense around the city area and more sparse in rural areas.

5.2. Running time

The running time of Algorithm 1 is compared with the exhaustive λ-optimal searches using the two data sets above. Figure 5 shows the results of the simulation data. The individual effect of each method in Sections 4.3 and 4.4 on the running time is also shown. Algorithm 1 with both methods runs fastest. The speedup is by at least a factor of 10 over the algorithm in Nikolova et al. (2006b). The speedup is due to the reduced number of λ-optimal searches. For the algorithm in Nikolova et al. (2006b), the number of λ-optimal searches gradually increases from 17 to 119 as the number of nodes increases from 100 to 10,000, whereas it increases from 5 to 7 for Algorithm 1.

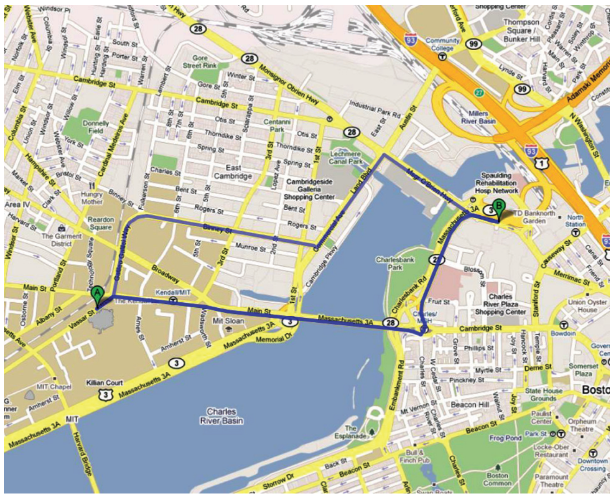


Fig. 6. Two major paths from MIT to the I-93 entrance. The upper path is denoted by 'Binney' and the lower path is denoted by 'Storrow'.

The running time of Algorithm 1 on the Delay Statistics Map for a route 144 km long is 14 s with five λ-optimal searches when the deadline is 3 hours. The same query took 178 s when we used the exhaustive λ-optimal search yielding 75 λ-optimal searches.

5.3. Path examples

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

5.4. Road experiment

5.4.1. Over-the-day path comparison We ran a driving experiment for two major paths from MIT to an interstate highway (I-93) entrance. The two paths are shown in Figure 6, where the origin is denoted as 'A' and the destination as 'B'. Two drivers drove each path on 4 April 2008, during the three time windows: morning rush hour (7:30am∼9:00am), mid-day hour (11:00am∼12:30pm), and evening rush hour (4:30pm∼6pm). The two drivers made four trips during the first two time windows and three trips for the last time window as shown in Figure 7.

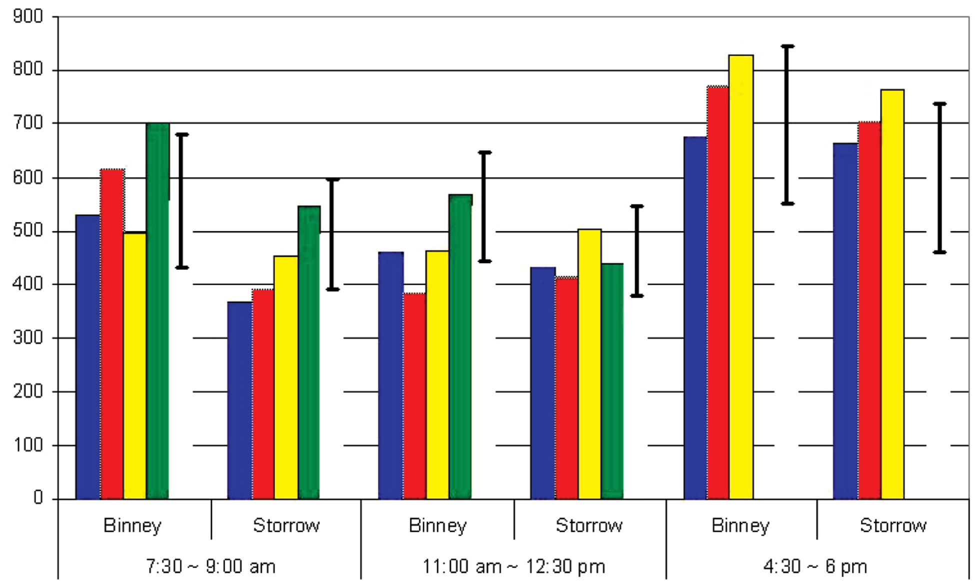


Fig. 7. The driving test data for 'Binney' and 'Storrow' on a single day (4 April 2008) by two test drivers. The bars represent each drive and the ranged bars represent our system's estimated travel time with the standard deviation from the mean (Y-axis unit: second).

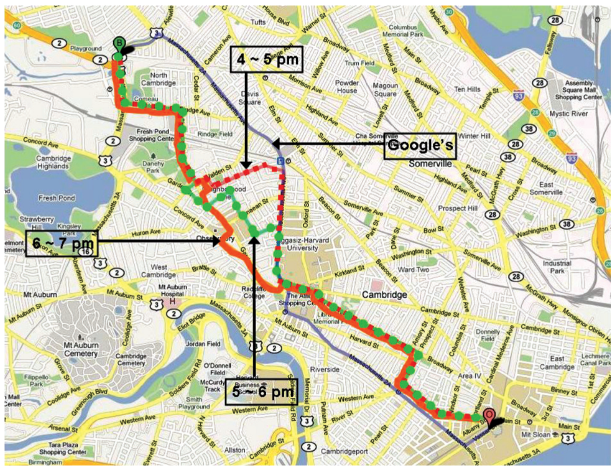


Fig. 8. The red square-dotted, green round-dotted, and orange solid lines indicate the minimum-expected-time paths for 4∼5pm, 5∼6pm, and 6∼7pm, respectively. The blue thin solid line indicates Google's path.

Figure 7 shows a large difference in travel time between morning, mid-day, and evening time travels. The measured travel times are mostly in the range of our system estimation. Our system estimated that path 'Storrow' has a smaller expected travel time than path 'Binney' in all three departure time groups, which is validated by the measured travel time data. Except for the second and third test drives of 11 am∼12:30 pm, the 'Storrow' path gave a smaller travel time than the 'Binney' path in all the other test drives.

5.4.2. Optimal paths for different departure times Our system returns different optimal paths for different departure times. Figure 8 shows different minimum-expected-time paths for different departure times on weekdays as well as Google's suggested path. Compared to Google's path taking Massachusetts Avenue, which is a major road where congestion is severe during the rush hour, our routes try to avoid this street and to find better alternative roads. Our system estimated that our time-dependent route is better than the Google path and we executed a real driving test by two human drivers where the route experienced by the two users was as estimated by our system.

On 18 and 21 November 2008 one driver drove our minimum-expected-time paths for 4∼5pm, 5∼6pm, and 6∼7pm during 4∼5pm, 5∼6pm, and 6∼7pm, respectively. Another driver drove Google's path from 4pm to 7pm. The two drivers made two trips per one hour windows. The result of driving on a Tuesday and on a Friday is shown in Figure 9. We can see that the travel time is usually longer on Fridays than Tuesdays, but the travel time for late evening is shorter on Fridays than on Tuesday. The peak is around 6∼7pm on Tuesdays, but it is around 5∼6pm on Fridays. This day-of-week specific tendency was well estimated by our system as shown in the second and third plots of Figure 9.

5.4.3. Overall path goodness Four different routes from MIT to Boston Logan airport were examined using taxi paths and human test driving. Figure 10 shows the four paths. The estimated mean of each path from 7am to 9pm was 872 s, 899 s, 816 s, and 795 s for route 1 (Memorial Drive, 6.9 km), route 2 (1st Street, 7.2 km), route 3 (Storrow Drive, 6.7 km), and route 4 (Cambridge Street, 6.2 km), respectively. The measured mean was 869 s, 895 s, 811 s, and 799 s. Thus, the estimated minimum expected time path, route 4, agrees with the measurement. Figure 11 gives the maximum probability path. The estimated probability is

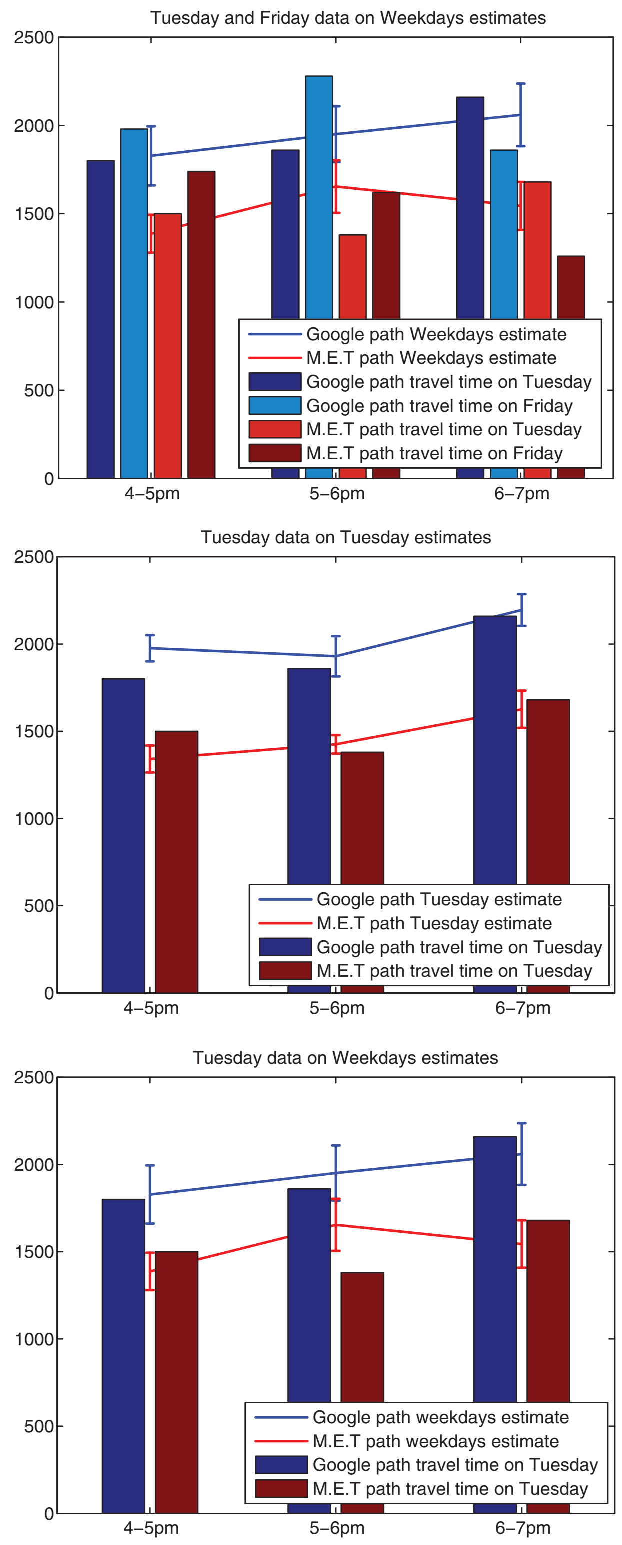


Fig. 9. The driving test data on a Tuesday (18 November 2008) and a Friday (21 November 2008) on top of our system's travel time estimation (Y-axis unit: second).

similar to the measured probability. From both estimation and measurement, for a deadline less than or equal to 12 min, route 3 is the best, but for a deadline larger than 12 min, route 4 is the best. We can observe that route 4, which is the minimum expected time path, is the worst path for a deadline of less than 12 min.

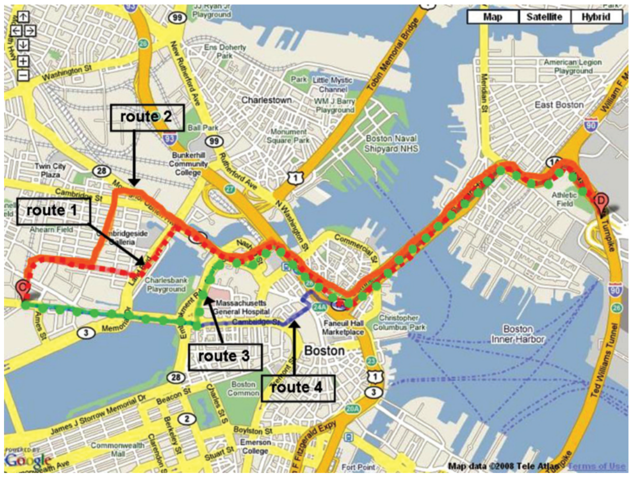


Fig. 10. Four major alternative routes from MIT (O) to Boston Logan Airport(D). Red square-dotted line : route 1, Orange solid line : route 2, Green round-dotted line : route 3, Blue thin solid line : route 4.

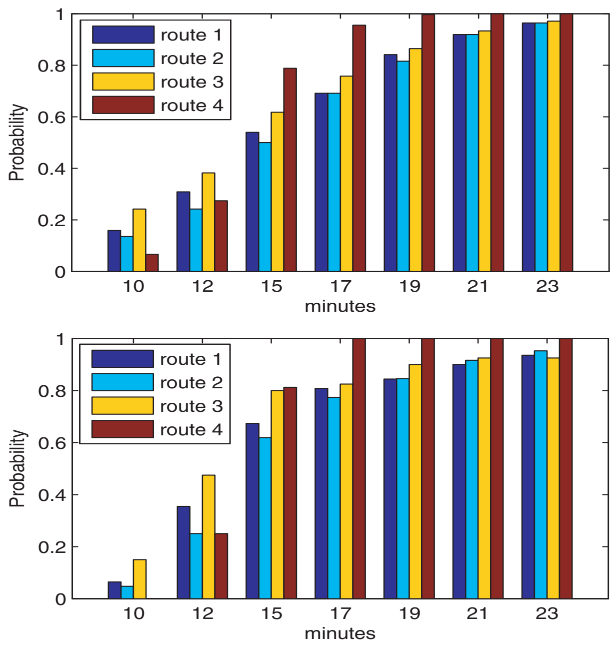


Fig. 11. The probability of arriving within a given deadline from an origin to a destination. (Top) Estimation. (Bottom) Measurement.

5.5. *Independent Gaussian assumption*

Our algorithms make an important assumption regarding independence between road segments. In this section we explore the independent Gaussian assumption using the taxi data. We believe that the scope in time and space of this data is statistically significant with respect to this study. To test

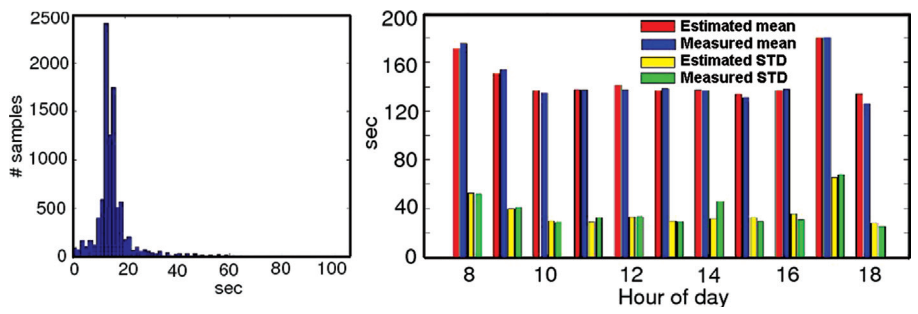


Fig. 12. (Left) Delay distribution for one segment. (Right) The comparison of mean and standard deviation between empirical measurement and estimation for various time slots.

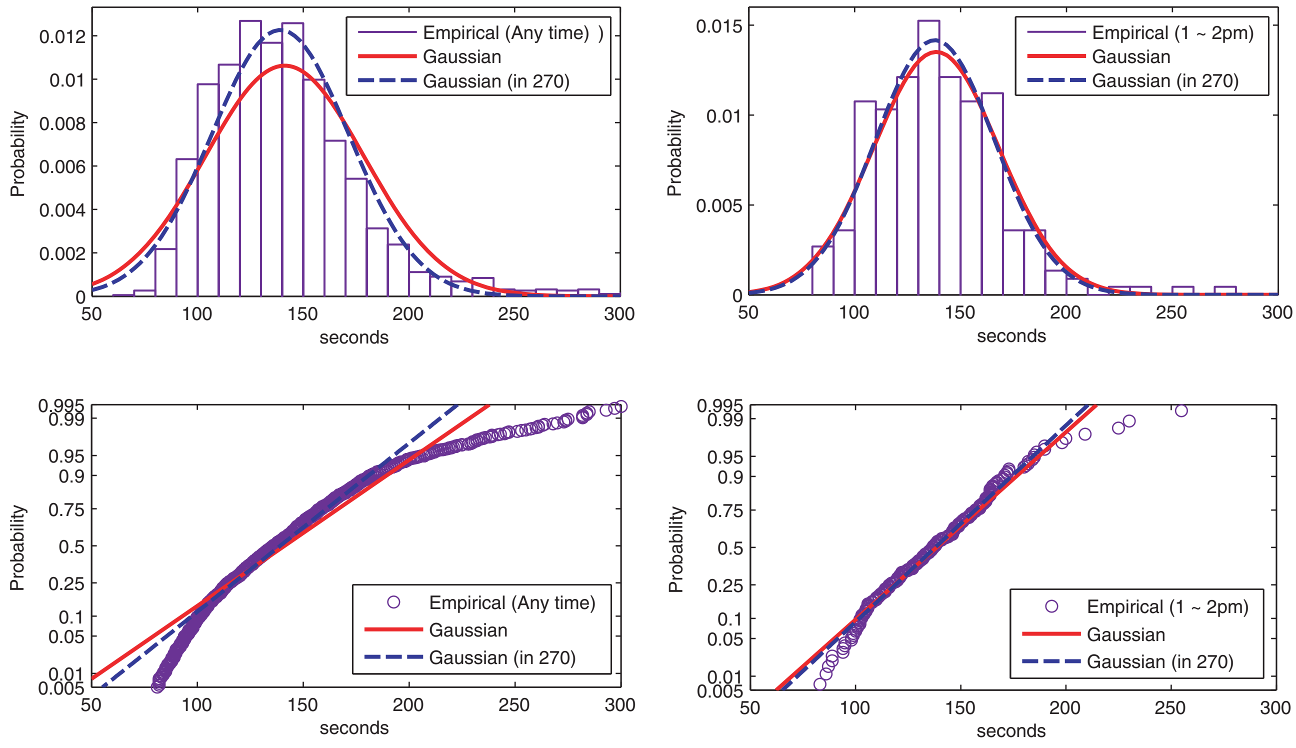


Fig. 13. ‘Empirical’ indicates the travel time measurement by driving, ‘Gaussian’ indicates the Gaussian fit for the entire data and ‘Gaussian(in 270)’ indicates the Gaussian fit using only the data in 270 s. (Top left) Histogram of empirical travel time data and Gaussian fits for any time for weekdays. (Bottom left) Probability comparison between empirical data and Gaussian fits for any time for weekdays, where the *Y*-axis was scaled to make the Gaussian CDF linear. (Right) Corresponding plots to the left plots for 1∼2pm for weekdays.

the independent Gaussian assumption, we identified a route with a large number of travel samples in the taxi trajectories database. We used a path from an origin to a destination, which is 1.12 km long and has five intersections and six road segments.

From Figure 12 (Right), we can observe that the empirical data and the estimation by independent Gaussian assumption is very close. More specifically, Figure 13 shows that the empirical data is very similar to the Gaussian distribution, especially in the probability interval from 0.05 to 0.95. Thus, our assumptions will hold well for the stochastic planning for reaching the destination with the probability in this range. The discrepancies observed over 0.95 and under 0.05 show the limitation of our algorithms due to our assumption. For example, as shown in Figure 13 (Bottom left), our system will estimate that the users can reach the goal with 99% probability if they leave 230 s before the deadline, but the empirical data shows that we will get only a 97% chance, and if users want a 99% guarantee they should leave about 270 s before the deadline. The discrepancy over 0.95 is caused by some unusual long delay, which might be due to unexpected events, construction work, or data gathering noise such as taxi drivers’ intentional stops or slow drives. The discrepancy below

0.05 is due to the Gaussian distribution spanning to a negative value whereas the travel time cannot be negative. We observe less discrepancy in the case of the right plots, which use only the data from 1∼2pm whereas the left plots are for entire hours. This result suggests that narrowing the data by conditions that affect the traffic delays, such as time of day, makes the delay distribution look more like an independent Gaussian distribution. Thus, in our ongoing research, we are investigating the proper conditions that constrain the traffic delays.

This observation provides some evidence that the independent Gaussian assumption used for Algorithm 1 holds for the taxi trajectories database, but more testing on road segments with more associated travel data is necessary. As the database grows every day, we plan to continue this validation.

6. Conclusion and future work

We developed efficient stochastic shortest-path algorithms that can be used for a practical traffic information system. We evaluated the system with actual measured travel time for selected routes, and observed that our system's optimal path and travel time estimates are close to reality.

In the future, we are interested in developing path planning algorithms for multiple users. We are also interested in improving our algorithms and system by considering dependencies of each road segment and by using better modeling of delay distributions. We are planning to consider time-dependent delay distributions in our future work. In addition, incorporating online information revealed as the vehicle travels, such as obstructed routes or congested routes, would be a great extension for our algorithms. We are currently extending the algorithms to integrate current traffic information with historical information to make more accurate estimations and to predict the future traffic conditions. Considering various conditions affecting traffic such as weather, construction work, and events is also part of our current plans. Finally, we plan to integrate this planning system with autonomous vehicles.

Acknowledgements

We thank PlanetTran for participating in the taxi data collection.

Funding

This work has been supported in part by NSF grants EFRI-0710252, IIS-0426838, and CNS-0931550, by the SWARMS and SMARTS MURI N00014-09-1-1051, and by a Samsung Scholarship. We are grateful for this support.

Notes

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

References

Bertsekas DP and Tsitsiklis JN (1991) An analysis of stochastic shortest path problems. *Mathematics of Operations Research* 16(3):580–595.

Carstensen, P (1983) The complexity of some problems in parametric linear and combinatorial programming. PhD thesis, Mathematics Department, University of Michigan, Ann Arbor, MI, January.

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

Chrobok R, Wahle J and Schreckenberg M (2001) Traffic forecast using simulations of large scale networks. In: *IEEE Conference on Intelligent Transportation Systems*, Oakland, CA, USA, August 2001.

Hull B, Bychkovsky V, Zhang Y, Chen K, Goraczko M, Miu AK, et al. (2006) CarTel: a distributed mobile sensor computing system. In: *4th ACM SenSys*, Boulder, CO, November 2006.

Lim S, Balakrishnan H, Gifford D, Madden S and Rus D (2008) 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.

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

Murthy I and Sarkar S (1997) Exact algorithms for the stochastic shortest path problem with a decreasing deadline utility function. *European Jounal of Operational Research* 103: 209–229.

Nikolova E, Brand M and Karger D (2006) Optimal route planning under uncertainty. In: *International Conference on Automated Planning and Scheduling*, Cumbria, UK / June 6–10, 2006.

Nikolova E, Kelner JA, Brand M and Mitzenmacher M (2006) Stochastic shortest paths via quasi-convex maximization. In: *ESA*, Zurich, Switzerland / Sep. 11–13, 2006.

Sanwal KK and Walrand J (1995) Vehicles as probes. Technical Report UCB-ITS-PWP-95-11, California Partners for Advanced Transit and Highways (PATH), January.

Schrank D and Lomax T (2007) The 2007 urban mobility report. Annual Report, Texas Transportation Institute, The Texas A&M University System, September.

Sigal CE, Pritsker AAB and Solberg JJ (1980) The stochastic shortest route problem. *Operations Research* 28(5): 1122–1129.

Wellman MP, Ford M and Larson K (1995) Path planning under time-dependent uncertainty. In: *11th Conference on Uncertainty in Artificial Intelligence*, August 1995, 532–539.

Xu H and Dailey DJ (1995) Real time highway traffic simulation and prediction using inductance loop data. In: *Vehicle Navigation and Information Systems Conference*, Seattle, WA, USA, July 1995.

Yoon J, Noble B and Liu M (2007) Surface street traffic estimation. In: *MobiSys '07: Proceedings of the 5th International Conference on Mobile Systems, Applications and services*, New York, NY, 2007. ACM, 220–232.