Providing Reliable Route Guidance using Chicago Data

Prepared by

Yu (Marco) Nie and Xing Wu
Department of Civil & Environmental Engineering
Northwestern University

Peter Nelson and John Dillenburg
Department of Computer Science, University of Illinois, Chicago

Technical Report #2009-001
Center for the Commercialization of the Innovative Transportation Technology

October 5, 2009

Contents

Chapter 1

Introduction

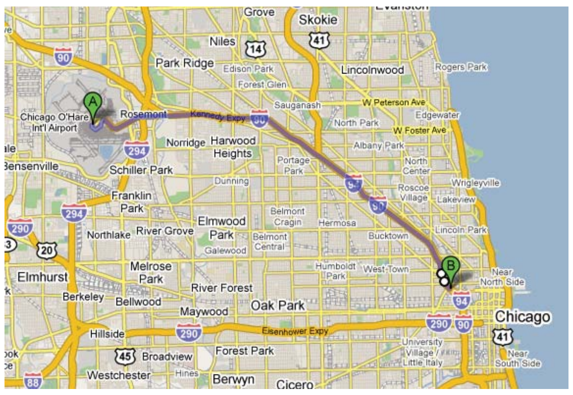
1.1 Background

Travel reliability is a critical performance dimension of transportation systems and services. It enables people and firms to make better use of available resources, including time, through effective personal and business activity scheduling. Shippers and freight carriers need predictable travel times to fulfill on-time deliveries and other commitments in order to remain competitive. The ability to arrive on-time with high reliability is imperative to emergency responders. However, urban transportation systems are affected by uncertainties of various sorts, which can be broadly classified as those affecting the supply of transportation (e.g., weather, accidents, natural and man-made disasters) and those associated with the demand for transportation (e.g., travel and activity behavior, special events). Taken individually or in combination, these factors could adversely affect and perturb the quality of transportation services. Travel behavior researchers have established that unanticipated long delays on highways typically produce much worse frustration among motorists than ”predictable” ones. The U.S. Federal Highway Administration (FHWA) estimates that 50-60% of congestion delays in most metropolitan areas are non-recurrent, and the percentage is even higher in smaller urban areas (FHWA 2000). To hedge against travel time fluctuations, travelers have to budget a sizable time buffer. In 1982, a 20-minute free-flow trip requires on average an extra 12-minute buffer time if on-time arrival is important (FHWA 2005). In contrast, the same trip would require 60% more buffer time in 2003. The reliability issue is likely to get still worse in the years to come, in light of limited capacity addition in the face of continuing growth in demand. Integrating travel reliability into transportation network analysis methods presents a pressing challenge that is of both theoretical and practical importance. This research addresses a particular aspect of this important subject, namely route guidance that respects the reliability requirement. The focus is to develop methods and procedures to implement *reliable* route guidance, and moreover, to demonstrate its utility.

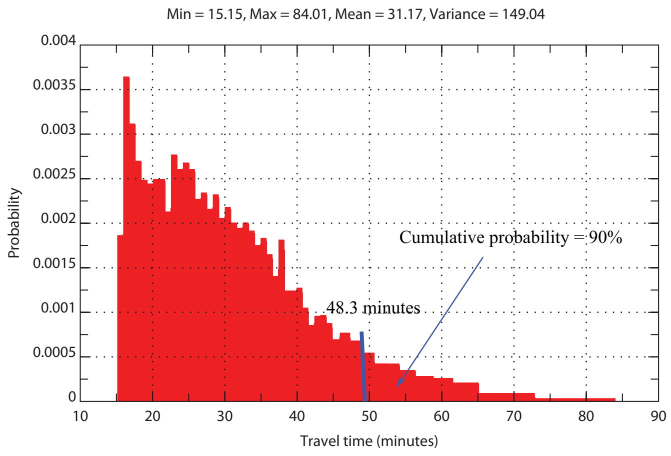
1.2 Route guidance and reliability

Motorists are becoming increasingly dependent on route guidance to plan unfamiliar trips. Just like a search engine can help internet users to locate useful information on web, a route guidance system finds best routes for motorists from a complex road system. As of today, many personal vehicles have built-in or adds-on GPS-based route guidance systems which can provide en-route guidance. Some of these equipments can even receive and make use of real-time traffic information. In the absence of such an in-vehicle unit, a priori driving directions (e.g. those provided by Internet-based map engines) are often used in trip planning.

Most existing route guidance systems assume that the road travel times are deterministic. When the stochastic nature of the system is acknowledged, the mean value is usually employed as the nominal travel time when computing optimal routes. However, our day-to-day experience suggest that not only travel time is random, but also a route with minimum mean travel time may be subject to large variances and therefore not always reliable. This is especially true in large metropolitan areas where random disruptions of various sorts consume a large portion of the total journey time. Figure 1.1 shows an example from Chicago area. The right panel in the figure displays the empirical distribution of travel times observed in weekday morning rush hour on a stretch of freeway that connects Chicago downtown to Ohare International Airport (shown in the left panel), the second busiest airport in the US. Note that the travel times vary from as low as about 15 minutes to as long as 80 minutes in that period. In light of the magnitude of the variance, it is not surprising that the travel time estimated by existing route guidance systems often turns out to be wildly inaccurate. Moreover, the figure also shows that if a traveler wish to capture the flight on time with a 90% chance, 48 minutes has



(a) Interstate 94/90 from Chicago (Ohio St.) to Ohare International Airport (source: Google Map)



(b) Travel Time Distribution for that corridor during morning rush hour (6-10 AM)

Figure 1.1: An illustration of travel time variances in the Chicago area

to be budgeted for travel, which is more than 50% more than the mean travel time (31 minutes). Existing route guidance systems neither allow users to incorporate the above reliability concerns into route choice, nor provide such information for recommended routes. Consequently, these systems essentially leave it to drivers to choose between running the risk of being late or budgeting a large buffer time, of which much is likely to be wasted. *Reliable route guidance* studied in this research addresses precisely these issues.

Reliable route guidance promises to enhance mobility and allow travelers to make better use of their time, in that it helps avoid overly conservative time budget. Reliable routing is also useful to freight carriers and parcel delivery firms whose trucks must move through peak-period traffic and work zones on a regular basis. Reliable route guidance allows these carriers to evaluate alternative routing plans for their fleets against the likelihood of on-time delivery, which is often an important performance index in the trucking industry.

In this research, the problem of generating reliable route guidance is modeled as the *reliable a priori shortest path problem* (RASP). The RASP problem aims to find a priori paths that are shortest to ensure a specified probability of on-time arrival. (Nie & Wu 2009*b*) showed that the RASP problem belongs to a class of multi-criteria shortest path problems, which rely on a dominance relationship to obtain Pareto-optimal solutions, that is, no travel time improvement associated with any on-time arrival probability can be further made without worsening those associated with other probability levels. Thus, the solution to the RASP problem is a reliable path set (instead of a single path) from which a motorist can pick the best corresponding to his / her desired origin / destination and the level of reliability.

Previous studies (Nie & Wu 2009*b*, Nie & Wu 2009*a*) have given the mathematical formulation for the RASP problem, examined the analytical properties and designed various solution algorithms. The goal of this research is to resolve the relevant implementation and deployment issues in order to move the techniques one step further to practice, and potentially commercialization. To these ends, three key issues are identified and addressed. The first is acquiring necessary data to prepare inputs for the RASP. Most important of all are road travel time distribution data, which are not directly available from existing traffic data collection and archiving practice and therefore have to be constructed from raw data. Second, the benefits of reliable routing over the conventional routing modes are demonstrated using real data. Finally, through comprehensive numerical experiments, this report verifies the feasibility of existing solution techniques in generating reliable route guidance on very large regional networks.

1.3 Organization

The organization of this report is as follows. Chapter 2 briefly reviews the literature on reliable routing guidance. Chapter 3 presents the mathematical formulation and solution algorithms for the reliable routing problem. Chapter 4 describes an object-oriented computer implementation of the RASP algorithm and introduces the graphic user interface of Chicago Travel Reliability (CTR). Chapter 5 provides an overview of the case study and describes how input data are obtained and processed, particulary the methods to be used where traffic data are not available. Chapter 6 reports and discusses experiment results from the case study. Chapter 7 concludes the study with a summary of findings.

Chapter 2

Literature

Route guidance algorithms direct vehicles from an origin to a destination along a path that are considered "optimal" one way or another. Depending on whether or not the guidance is coordinated by a central control unit, the algorithms can be classified as "centralized" or "decentralized". They can also be labeled as "adaptive" or "a prior", according to whether or not en-route re-routing is allowed. Two other factors that are often used in classification are dynamics (i.e., if travel time varies over time of day) and uncertainties (i.e., if travel time is random). This research considers decentralized, *a prior* route guidance for stochastic and static networks [1]. The focus is to incorporate travel reliability as an integrated objective of route guidance. By *static*, we mean that the travel time distributions remain constant within each routing process. We have to restrict to the static case not because of methodological limitations [2], but rather due to data availability. The static label does not exclude, however, the possibility of changing travel time distributions according to time-of-day from one routing process to another. As shown in the case study, reliable routes generated for morning rush hour are likely to be different from those for evening peak period.

When uncertainties are concerned, "optimal" routing has many different meanings. A classic definition considers a routing strategy optimal if it incurs the least expected travel time (LET). Another common definition of optimality in stochastic routing has to do with reliability, recognizing that a LET route (or policy) may be subject to high risks

[1] Corresponding "adaptive" problems are related to "a prior" counterparts, and are usually simpler to solve.

[2] Note that both Miller-Hooks (1997) and Nie & Wu (2009*b*) deal with "dynamic" version of the problem.

and therefore is not desirable to a *risk averse* traveler.

2.1 LET problems

Finding LET paths is trivial when random link travel times are independently distributed and do not vary over time. Non-trivial LET problems are revealed when these assumptions are relaxed. An important variant is the adaptive LET problem with recourse in which traversal time on a link will become known and deterministic upon the arrival of its tail (starting) node (e.g. Croucher 1978, Andreatta & Romeo 1988, Polychronopoulos & Tsitsiklis 1996, Cheung 1998, Gao & Chabini 2006, Waller & Ziliaskopoulos 2002, Provan 2003). Correlations between link travel times are explored in Waller & Ziliaskopoulos (2002) and Fan, Kalaba & Moore (2005*b*). Both studies consider only the adaptive LET problem and assume the knowledge of transitional probabilities which depict spatial correlations between adjacent links. Another class of problems assume that link travel time distributions are conditional on the arrival time at link entrance. Both *a priori* and *adaptive* variants of the problem have been studied (e.g., Hall 1986, Fu & Rilett 1998, Miller-Hooks & Mahmassani 2000, Miller-Hooks 2001, Fu 2001). Due to the absence of the Markovian property, finding LET paths over stochastic and time-dependent networks is difficult. Existing solution procedures are either pure heuristics (Fu & Rilett 1998) or non-deterministic polynomial (Miller-Hooks & Mahmassani 2000).

2.2 Reliability-based problems

It has been recognized that the LET path may be subject to high risk since it overlooks travel time variances. This concern gives rise to the reliability-based stochastic routing problem. Reliability-based stochastic routing has been studied extensively, with the majority of the literature focused on *a priori* path problems.

2.2.1 Reliable *a priori* shortest path problem

In his seminal work, Frank (1969) defines the optimal path as the one that maximizes the probability of realizing a travel time equal to or less than a given threshold. The distributions of all link travel times are continuous, and the convolution integral is calculated using characteristic functions. The shortest paths are identified through a pairwise comparison in a set of enumerated paths. Since this method requires path enumeration, applying it in large networks is difficult.

Mirchandani (1976) presented a recursive algorithm to solve a discrete version of the Frank's problem (1969). However, the algorithm is suitable only for very small instances since it requires to enumerate not only all paths but also all travel time possibilities for each path through a network expansion.

Sigal, Alan, Pritsker & Solberg (1980) suggested using the probability of being the shortest path as an index to define the optimality. For path l, the optimality index R_l is defined as:

$$R_l = P(Y_l \leq Y_1, Y_l \leq Y_2, \cdots, Y_l \leq Y_k, \cdots, Y_l \leq Y_m) \tag{2.1}$$

where $k = 1, \cdots, m$ and $k \neq l$. Equation (2.1) is transformed to a multi-integral equation. To solve that multi-integral equation, a set of path-independent links, i.e., a set of links such that no two links in the set are on the same paths, are needed. Again, it is difficult to determine such a set unless all paths are enumerated.

The expected utility theory of von Neumann & Morgenstern (1967) has also been used to define path optimality. In Loui (1983), a path weight (cost) is defined as

$$\pi_k^{rs} = \sum_{\forall ij \text{ s.t. } \delta_{ij}^k = 1} c_{ij} \tag{2.2}$$

where c_{ij} is the weight (cost) on link ij, k^{rs} refers to a path from node r to node s; and $\delta_{ij}^k = 1$ implies that link ij is traversed by path k^{rs}, otherwise, $\delta_{ij}^k = 0$.

Given a utility function $u(x)$ that is monotonically decreasing in x, path k^{rs} is preferred to path l^{rs} if and only if $u(\pi_k^{rs}) > u(\pi_l^{rs})$. Therefore the optimal path is defined as

$$k{*}^{rs} \equiv \arg \max_{k^{rs} \in K^{rs}} [u(\pi_k^{rs}))] \tag{2.3}$$

where K^{rs} is the set of all paths between OD pair rs. Loui (1983) showed that the Bellman's principle of optimality applies to affine or exponential utility functions. This restriction was independently observed in Eiger, Mirchandani & Soroush (1985). For a general polynomial and monotonic utility function, Loui's expected-utility problem can be reduced to a class of bi-criterion shortest path problems that involves mean and variance of a path. In effect, this reduction allows one to tradeoff the expected value and variance (reliability) using generalized DP (see, e.g., Carraway, Morin & Moskowitz 1990) based on pareto optimality (or dominance relationship). More general nonlinear utility functions may be approximated by piecewise linear functions, see, e.g. (Murthy & Sarkar 1996, Murthy & Sarkar 1998), who also proposed a few efficient solution procedures based on relaxation. The mean-variance tradeoff has been treated in different ways. For example, Sivakumar & Batta (1994) adds an extra constraint into the shortest path problem to ensure that the identified LET paths have a variance smaller than a benchmark. In Sen, Pillai, Joshi & Rathi (2001), the objective function of stochastic routing becomes a parametric linear combination of mean and variance. In either case, DP cannot be applied. Instead, nonlinear or integer programming solution techniques must be used.

Stochastic routing has also been discussed in the context of robust optimization, that is, a path is optimal if its worst-case travel time is the minimum (Yu & Yang 1998, Montemanni & Gambardella 2004). However, such robust routing problems are NP-hard even under restrictive assumptions (Yu & Yang 1998).

Bard & Bennett (1991) defined the optimal path as the one that maximizes the expected utility in a stochastic acyclic network. Compared with the study of Loui (1983) where utility functions have to be polynomial and monotonic, Bard & Bennett (1991) only require that the utility function to be non-linear and monotonic. In order to solve the global optimal path, all paths have to be enumerated. To improve computational performance, the authors proposed to reduce the network size using first order stochastic dominance (FSD). Specifically, if a path is dominated by other(s) in the first order, the links on that path can be eliminated. Since FSD is only a necessary condition, other criteria for link elimination were also proposed. To apply FSD, all link travel time distributions are discretized by N support points, so $x_i < y_i, \forall i = 1, \cdots, N$ implies that X dominates Y, i.e., $X \succ_1 Y$. In fact, the FSD condition is relaxed in Bard & Bennett (1991) as ”post median stochastic dominance”, in which only the points on the tail of CDF (instead of the entire CDF) are checked. Bard & Bennett (1991) observed that 90% of paths in an acyclic network can be eliminated after link reduction. Therefore, finding optimal path through enumeration is easier in the reduced network. The algorithm, however, can only be applied to acyclic networks, and thus it is not suitable for large real networks.

Miller-Hooks & Mahmassani (1998) defined the optimal path in a stochastic and time-varying network as the one that realizes the least possible travel time. Miller-Hooks (1997) and Miller-Hooks & Mahmassani (2003) explored three other types of optimality: 1) deterministic dominance, 2) the first-order stochastic dominance, and 3) expected value dominance. Label-correcting algorithms are proposed to find non-dominated paths under the stochastic dominance rules. Recognizing that the exact algorithm does not have a polynomial bound, heuristics are considered (Miller-Hooks 1997) which attempt to limit the size of the retained non-dominant paths by a predetermined number. As noted in Miller-Hooks (1997) (Chapter 5), however, these heuristics may not identify any non-dominant paths.

Reliability has also been defined using the concept of connectivity (Chen, Bell, Wang & Bogenberger 2006, Kaparias, Bell, Chen & Bogenberger 2007). This approach models reliability as the probability that the travel time on a link is greater than a threshold. Accordingly, the reliability on a path is the product of link reliability (assuming independent distributions). A software tool known as ICNavS was developed based on this approach (Kaparias et al. 2007).

2.2.2 Reliable adaptive routing problem

Another class of reliable routing problem has to do with adaptive policy instead of *a priori* path. Bander & White (2002) studied the reliable adaptive routing problem by imposing penalty on early and late arrival for a pre-defined arrival time. The objective is to determine a policy Π^* such that the expected penalty under Π^* is always less than the expected penalty under any other policy Π. The problem can be solved via dynamic programming similar to that in Miller-Hooks & Mahmassani (2000). However it requires to enumerate all policies. Bander & White (2002) then proposed a heuristic algorithm AO* to solve the problem. AO* is a heuristic search technique that is shown to be more computationally efficient than dynamic programming when the lower bound on the value function are available. However, AO* can only be applied to acyclic network.

Fan, Kalaba & Moore (2005*a*) studied an optimal routing problem known as stochastic on time arrival (SOTA) problem. SOTA attempts to find an optimal adaptive routing policy that maximizes the probability of arrival on time given a travel time budget. Denote $u_i(t)$ as the maximum probability of arriving at the destination from node i within a given time budget t. Suppose j is the next node to traverse, and ω is the travel time on link ij, s is the destination, and T is the time budget given at the origin node r. Fan et al. (2005*a*) formulated SOTA as a system of integral equations as follows (with Equation (2.5) giving the boundary condition).

$$
\begin{aligned}
u_i(t) &= \max_{j \neq i} \int_0^t p_{ij}(\omega) u_j(t-\omega) d\omega, \forall i \in N, i \neq S, 0 \leq t \leq T && (2.4) \\
u_s(t) &= 1, 0 \leq t \leq T && (2.5)
\end{aligned}
$$

Note this formulation assumes that the link travel time is an independent random variable. Nie & Fan (2006) proposed a discrete version of the SOTA problem and a solution algorithm called ”increasing order of time budget (IOTB)”.

2.3 Prior work

This research is based on the prior work of Nie and Wu (2009*b*, 2009*a*, 2009), which defines the objective of routing as maximizing the probability of arriving on-time and formulates the problem using general dynamic programming. This definition of optimality is identical to that of Frank (1969) and closely related to the first-order stochastic dominance of Miller-Hooks & Mahmassani (2003). The definition is adopted because it is intuitively addresses drivers' concern about travel reliability. Solving the above reliable routing problem on real networks has been considered impractical because it requires path enumeration. Nevertheless, this research is built on the premises that recent advances in algorithmic development and computer technology warrants a fresh look at this once “intractable” problem. The formulation of the problem and its solution algorithms are presented in Chapter 4.

Chapter 3

Formulation and Algorithm

In this section, we introduce the mathematical formulation and solution algorithm for the *reliable a priori shortest path (RASP) problem*, which aims to find *a priori* paths that are shortest to ensure a specified probability of on-time arrival. We first introduce notation in Section 3.1.

3.1 Notation

Consider a directed and connected network $G(\mathcal{N}, \mathcal{A}, \mathcal{P})$ consisting of a set of nodes $\mathcal{N}$ with the number of nodes $|\mathcal{N}| = n$, a set of links $\mathcal{A}$ with the number of links ($|\mathcal{A}| = m$), and a probability distribution $\mathcal{P}$ describing the statistics of link travel times. The analysis time period is set to $[0, T]$. Let the destination of routing be node s and the desirable arrival time be aligned with the end of the analysis period T. The traversal times on different links (denoted as c_{ij}) are assumed to be independent random variables, each of which follows a random distribution with a probability density function $p_{ij}(\cdot)$ and let $F_{ij}(\cdot)$ be the cumulative density function (CDF) of c_{ij}. The link traversal times are assumed to be independent in our numerical tests.

The travel time on path k^{rs} (which connects node r and the destination s) is denoted as π_k^{rs}. All paths that connect node r and s form a path set K^{rs}. Finally, let $u_k^{rs}(\cdot)$ denote the CDF of π_k^{rs}. Therefore, $u_k^{rs}(b)$ represents the maximum probability of arriving at destination s through path k^{rs} no later than T, departing r with a time budget b.

3.2 A RASP formulation derived from stochastic dominance

The RASP problem is highly related to stochastic dominance (SD), a theory that has been extensively used in finance and economics to compare random quantities. We briefly review the SD theory in the following and reveal how it defines the optimality for the RASP problem.

Conventionally, the first order stochastic dominance is defined as follows.

Definition 3.1 (FSD $\succ^1$) *A random variable X dominates another random variable Y in the first order, denoted as $X \succ^1 Y$ if $F_X(t) \leq F_Y(t) \forall t$ and at least one strict inequality.*

With Definition 3.1, Theorem 3.1 is a well-known result (see e.g. Levy & Hanoch 1970).

Theorem 3.1 *X dominates Y in the first order if and only if $E[U(X)] \geq E[U(Y)]$, for any nondecreasing utility function $U(\cdot)$, i.e., $U' \geq 0$.*

Definition 3.1 is based on the circumstance that decision-makers are always better off with more quantities of the random variable of interest. That is, their utility function is non-decreasing with respect to the quantity. Clearly, this is not always the case. For example, travelers usually prefer shorter travel times to longer ones. In other words, travel time is considered as disutility instead of utility. When random variables are associated with disutility, the stochastic dominance has to be re-defined as follows.

Definition 3.2 (FSD by disutility $\succ_1$) *A random variable X dominates another random variable Y in the first order by disutility, denoted as $X \succ_1 Y$, if $F_X(t) \geq F_Y(t), \forall t$ and at least one strict inequality.*

Theorem 3.1 still holds in Definition 3.2, yet the utility functions now have to be non-increasing instead of non-decreasing.

We are now ready to compare two paths, whose disutility is measured by its travel time, using the dominance relationship.

Definition 3.3 (Path dominance) *Let π_k^{rs} and π_l^{rs} be the random travel times on path k^{rs} and l^{rs} respectively. If $\pi_k^{rs} \succ_1 \pi_l^{rs}$ then path k^{rs} is said to dominate path l^{rs} in the first order.*

Definition 3.4 (FSD-admissible paths) *A path k^{rs} is FSD-admissible if there $\exists$ no path in K^{rs} that can dominate path k^{rs} in the first order, where K^{rs} is the set of all paths between node r and node s.*

Denote Γ_1^{rs} as the set of the FSD-admissible paths from node r to destination s. FSD-admissible paths are also called as **non-dominant** paths in the literature (Miller-Hooks 1997, Miller-Hooks & Mahmassani 1998, Miller-Hooks & Mahmassani 2003, Nie & Wu 2009*b*).

Further we have the following definition.

Definition 3.5 (Pareto-optimal paths) *An FSD-admissible path $\bar{k}^{rs}$ is said to be Pareto-optimal path, if for at least one time budget b no other path can provide a higher on-time arrival probability.*

Introduce

$$u^{rs}(b) = \max_{\forall k^{rs} \in \Gamma_1^{rs}} \left\{ u_k^{rs}(b) \right\} \tag{3.1}$$

as the Pareto frontier. That is, if a path k^{rs} is Pareto-optimal, then $\overline{k}^{rs} = \text{argmax}[u_k^{rs}(b), \forall k^{rs} \in K^{rs}]$ at b. Given a time budget b, path $\overline{k}^{rs}$ ensures the highest on-time arrival probability among all paths from node r to node s.

The above analysis employs the cumulative density function $u_k^{rs}(\cdot)$ which gives an on-time arrival probability α based on a specific travel time budget b. Conversely, we can define the optimality with a given on-time arrival probability. Denote $v_k^{rs}(\cdot)$ as the inverse function of $u_k^{rs}(\cdot)$, and an alternative Pareto frontier and the Pareto-optimal path $\overline{k}^{rs}$ are respectively defined as

$$v^{rs}(\alpha) = \min_{\forall k^{rs} \in \Gamma_1^{rs}} \left\{ v_k^{is}(\alpha) \right\} \tag{3.2}$$

$$\bar{k}^{rs} = \text{argmin} \left[v_k^{rs}(\alpha), \forall k^{rs} \in K^{rs} \right] \tag{3.3}$$

According to Equations (3.2) and (3.3), the optimal solution to the RASP problem is the minimum time budget and the associated path, corresponding to a given probability.

The following proposition is obvious, given $u_k^{rs}(v_k^{rs}(\alpha)) = \alpha$, $\forall \alpha, \forall k^{rs} \in K^{rs}, \forall rs$.

Proposition 3.1 *If a path $\bar{k}^{rs}$ is Pareto-optimal, and $\bar{k}^{rs} = \text{argmin}\left[v_k^{rs}(\alpha_0), \forall k^{rs} \in K^{rs} \right]$, then $\exists b_0$ such that $u_k^{rs}(b_0) = u^{rs}(b_0) = \alpha_0$. On the other hand, $\bar{k}^{rs} = \text{argmax}\left[u^{rs}(b_0) \right]$, and $v_k^{rs}(\alpha_0) = v^{rs}(\alpha_0) = b_0$.*

Proposition 3.1 means if a path gives the highest on-time arrival probability for a time budget, it implies that the travel time budget is minimum for that on-time arrival probability.

Proposition 3.2 *An FSD-admissible path is not necessarily Pareto-optimal.*

Figure 3.1 shows the cumulative density functions $u_k^{rs}(\cdot), k = 1, 2, 3$ for three paths and the Pareto frontier u^{rs}. Given there are only three paths between OD pair rs, it shows that path 3 is an FSD-admissible path because no path can dominate it in the first order. However, $u_3^{rs}(\cdot)$ (solid line) does not contribute to the Pareto frontier. Therefore path 3 is not a Pareto-optimal path.

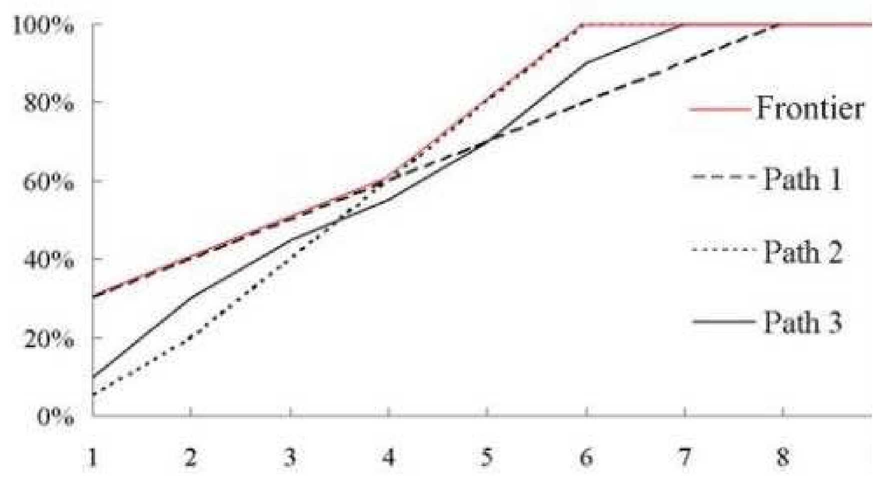


Figure 3.1: An FSD-admissible path that is not Pareto-optimal

We now discuss two important properties of FSD-admissible paths. Before that, we note that $u_k^{is}(b)$ (the CDF of random path travel time π_k^{rs}) can be recursively calculated by

$$u_k^{is}(b) = \int_0^b u_k^{js}(b-w)p_{ij}(w)dw \tag{3.4}$$

if the distribution of link traversal time is continuous and independent (Frank 1969).

Proposition 3.3 *Subpaths of FSD-admissible paths must also be FSD-admissible paths.*

Proof: Let $k^{is} \in \Gamma_1^{is}$. Suppose one of its subpath of k^{js} ($k^{is} = k^{js} \diamond ij$) is not FSD or SSD-admissible, then there must exist a path l^{js} such that $l^{js} \succ_1 k^{js}$. Recalling

$$u_l^{is}(b) = \int_0^b u_l^{js}(b-w)p_{ij}(w)dw$$

Therefore given $l^{js} \succ_1 k^{js}$, we have $u_l^{js}(b) \geq u_k^{js}(b), \forall b$, then

$$u_l^{is}(b) - u_k^{is}(b) = \int_0^b [u_l^{js}(b-w) - u_k^{js}(b-w)] p_{ij}(w) dw \geq 0, \quad \forall b$$

Therefore it concludes that $l^{is} \succ_1 k^{is}$. It contradicts with the assumption that k^{is} is FSD-admissible. □

Proposition 3.4 *FSD-admissible paths must not contain any cycle.*

Proof: Suppose, without loss of generality, that path k^{rs} and l^{rs} are either FSD or SSD-admissible. l^{rs} contains one and only one cycle starting at node i, while k^{rs} is acyclic. Then according Equation 3.4, assuming the cycle from node i to node i is reduced to a link $\widetilde{ii}$, u_l^{is} is calculated as

$$u_l^{is} = \int_0^b u_k^{is}(b-w) p_{\widetilde{ii}}(w) dw$$

Then we have $u_l^{is}(b) \leq \int_0^b u_k^{is}(b) p_{\widetilde{ii}}(w) dw = u_k^{is}(b) \int_0^b p_{\widetilde{ii}}(w) dw \leq u_k^{is}(b) \leq u_k^{is}(b), \forall b.$

The first inequality is due to the monotonicity of CDF. The second inequality holds because $p_{\widetilde{ii}}(\cdot)$ is a probability density function, $\int_0^\infty p_{\widetilde{ii}}(w) dw = 1$, and $b < \infty$, so $\int_0^b p_{\widetilde{ii}}(w) dw < 1$. Therefore, $u_k^{is}(b) \geq u_l^{is}(b), \forall b$. It means that $k^{is} \succ_1 l^{is}$ (see Definition 3.3). Therefore l^{rs} has no chance to be an FSD-admissible path. □

Using the result of Proposition 3.3, the RASP problem can be formulated as the following general dynamic programming program.

$$\text{Find } \Gamma_1^{is}, \forall i \text{ such that } \Gamma_1^{is} = \gamma_{\succ}^1(k^{is} = k^{js} \diamond ij | k^{js} \in \Gamma_1^{js}, \forall ij \in \mathcal{A}), \forall i \neq s;\ \Gamma_1^{ss} = 0^{ss} \tag{3.5}$$

where $k^{ij} \diamond ij$ extends path k^{js} along link ij; $\gamma_{\succ}^1(K^{rs})$ represents the operations which retrieves SD-admissible paths from a path set K^{rs} using Definition 3.2; 0^{ss} is a dummy path representing the boundary condition.

3.3 Solution techniques

Solving the RASP problem (3.5) involves two main operations: iteratively constructing and storing admissible paths, and evaluating their travel time distributions. Central to either operation is how to discretize the underlying problem. We shall first introduce the basic algorithmic concept using a simple and easy-to-implement discretization scheme, and then discuss more sophisticated approaches.

3.3.1 Basic discretization scheme

Although the determination of stochastic dominance has to use a finite set of discrete points on the CDF, the convolution integral (3.4) can be evaluated in the continuous space. For instance, Fan et al. (2005*a*) suggest using Laplace Transformation to perform a similar integral. Their method makes use of the fact that the Laplace Transformation of a convolution equals the product of the individual transforms, that is

$$U(s) = F(s).G(s),\ F(s) = \int_0^\infty e^{-st} f(t)dt,\ G(s) = \int_0^\infty e^{-st} g(t)dt,\ U(s) = \int_0^\infty e^{-st} u(t)dt \tag{3.6}$$

where

$$u(t) = \int_0^t f(t-w)g(w)dw$$

The evaluation of a convolution is divided into three steps. Firstly, both functions (in Equation(3.4), they are $u_k^{is}(\cdot)$ and $p_{ij}(\cdot)$) are transformed and numerically integrated for a set of discrete s. The second step calculates the convolution of the transformed functions which turns out a point-to-point multiplication. Finally, the resulting function is reverted back to the original domain. The last step involves solving a linear system $U = Vu$ in which U and u are vectors of $U(s)$ and $u(t)$ evaluated at discrete points s and t, respectively, and V is a Vandermonde matrix. We did not adopt this method in this research because the resulting Vandermonde matrix is usually ill-conditioned and the inverse operation is therefore unstable.

Let $B = L\phi$ be the largest travel time value in consideration, where ϕ is the length of time unit. The time budget is treated in a discrete manner, i.e., $b = 0, \phi, 2\phi, \cdots, L\phi$. For any link ij, the probability mass function $P_{ij}(b)$ of c_{ij} may be obtained from its probability density function $p_{ij}(b)$ as follows:

$$P_{ij}(b) = \begin{cases} \int_b^{b+\phi} p_{ij}(w)dw & b = 0, \phi, ..., (L-1)\phi \\ \int_b^\infty p_{ij}(w)dw & b = L\phi \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

The above definition assumes that supporting points of any c_{ij} must take a non-negative multiple of ϕ. This assumption helps simplify the evaluation of convolution integral. Also note that for any discrete $b > L\phi$, the probability mass is assumed to be zero. Then the convolution integral (see Equation (3.4)) is replaced with a finite sum as follows:

$$u_k^{is}(b) = \sum_{0}^{b} u_k^{js}(b-\phi)P_{ij}(\phi),\ \forall b = 0, \phi, \cdots, L\phi \tag{3.8}$$

If a continuous link distribution p_{ij} is discretized to P_{ij} according to Equation (3.7), and then u_k^{is} is calculated from u_k^{js} and P_{ij} following Equation (3.8), we call this discrete approximation method as the ”*b-discrete*” method.

3.3.2 A label-correcting algorithm

We now describe a label correcting algorithm to search for all FSD-admissible paths. The description is based on (Nie & Wu 2009*b*), although an earlier and slightly different version has been studied in (Miller-Hooks 1997). The algorithm is named FSD-LC, with FSD stands for first-order stochastic dominance, and LC stands for label-correcting. The following definitions are used in the algorithm description: a list of candidate paths is denoted using Q; $\omega(\cdot)$ is a subpath operator used to track paths (e.g., $\omega(k^{is}) = k^{js}$ such that $k^{is} = k^{js} \diamond ij$); Ω_{is} denotes a tentative set of FSD-admissible paths.

Algorithm FSD-LC

Step 0 Initialize. Set $Q = \emptyset$, $u^{is}(b) = 0, \forall b = 0, \phi, .., L\phi, \forall i \neq s, \Omega_{is} = \emptyset, \forall i$. For the destination node s, set $u^{ss}(b) = 1, \forall b = 0, \phi, ..., L\phi$. Create a path from s to itself, 0^{ss} and set $u_0^{ss}(b) = 1, \forall b = 0, \phi, ...L\phi$. Let $Q = Q \cup \{0^{ss}\}$.

Step 1 Check optimality. if $Q = \emptyset$, terminate the procedure, the optimal solution is found; otherwise proceed to Step 2.

Step 2 Take the first path k^{js} stored in Q and scan every incoming link ij of node j.

step 2.1 If all links ij have been scanned, go to Step 1; otherwise, take the next link $ij \in \mathcal{A}$.

step 2.2 Check whether path k^{js} has already traversed node i. If yes, go back to step 2.1; otherwise, proceed to step 2.3.

step 2.3 Set $l = |\Gamma_1^{is}| + 1$, create a new path l^{is}, calculate

$$u_l^{is}(b) = \sum_{h=0}^{b} u_k^{js}(b-h)P_{ij}(h), \forall b = 0, \phi, ..., L\phi \tag{3.9}$$

step 2.4 if $\Gamma_1^{is} = \emptyset$, set $\Gamma_1^{is} = l^{is}, u^{is}(b) = u_l^{is}(b), \bar{k}^{is}(b) = l^{is}, \forall b = 0, \phi, ..., L\phi$. Update $\sigma(l^{is}) = L + 1, \omega(l^{is}) = k^{js}$;

otherwise, call Procedure FSD-CHECK. If l^{is} is not FSD-admissible, go to step 2.1; otherwise, set $\omega(l^{is}) = k^{js}$ and update $Q = Q \cup \{l^{is}\}$. Go to step 2.1.

Procedure FSD-CHECK

Inputs: a new path l^{is}, a set of FSD-admissible paths Γ_1^{is}, as well as the associated Pareto frontier $u^{is}(\cdot)$.

Return: a boolean value LR indicating whether or not l^{is} is FSD-admissible, and updated u^{is} and Γ_1^{is}.

Step 0 set LR = TRUE, set $\sigma(l^{is}) = 0$, set $Q' = \emptyset$ (Q' is the set of paths that are currently FSD-admissible but have a zero degree of strong dominance).

Step 1 Update Pareto frontier and identify Q.

```
for each $b = 0, \phi, ..., L\phi$ do
  set $k^{is} = \bar{k}^{is}(b)$.
  If $u_l^{is}(b) > u^{is}(b)$
    update $u^{is}(b) = u_l^{is}(b), \bar{k}^{is}(b) = l^{is}, \sigma(l^{is}) = \sigma(l^{is}) + 1, \sigma(k^{is}) = \sigma(k^{is}) - 1$
      If $\sigma(k^{is}) = 0$, set $Q' = Q' \cup \{k^{is}\}$.
  end if
end for
```

Step 2 Update the set of FSD-admissible paths.

```
while LR = TRUE and $Q'$ is not empty, do
  take path $k^{is}$ out of $Q'$, set $n_l = 0, n_e = 0, n_g = 0$.
  for $b = 0, \phi, ..., L\phi$ and if ($n_l = 0$ or $n_g = 0$) do
    if $u_l^{is}(b) > u_k^{is}(b)$, set $n_g = n_g + 1$; else if $u_l^{is}(b) = u_k^{is}(b)$, $n_e = n_e + 1$; else,
$n_l = n_l + 1$.
```

```
        end for
        if $n_l = 0$, set LR = FALSE; else if $n_g = 0$, set $\Gamma_1^{is} = \Gamma_1^{is} / \{k^{is}\}$.
    end while
```

Step 3 If LR = TRUE, set $\Gamma_1^{is} = \Gamma_1^{is} \cup \{l^{is}\}$. return LR.

The following remarks are in order.

Remark 1: While dealing with different problems, Algorithm FSD-LC is conceptually similar to the EV algorithm of Miller-Hooks & Mahmassani (2000), in which a non-dominance relationship is defined with respect to departure times instead of time budgets. However, FSD-LC promises to reduce the amount of work required to carry out the dominance check (although the strategy does not improve the worst-case scenario).

Remark 2: For each node i, Algorithm FSD-LC needs to store u^{is} and $\bar{k}^{is}$ - both are vectors of length $L+1$. Moveover, a vector u_k^{is} (length $L+1$) must be stored for each path $k^{is} \in \Gamma_1^{is}$.

To examine the complexity of Algorithm FSD-LC, the following proposition is needed.

Proposition 3.5 *In Algorithm FSD-LC, a scanned path will never reenter the candidate list Q.*

Proof: Note that a path enters Q only at initialization (Step 0) or in step 2.4. In the latter situation, the path is always newly generated in step 2.3, on top of existing set of FSD-admissible paths Γ_1^{is}. □

It follows from the lemma that Algorithm FSD-LC must terminate after finite steps since the number of acyclic paths in a directed network is finite. This is formally stated as below.

Theorem 3.2 *Algorithm FSD-LC terminates after a finite number of steps and yields a set of FSD-admissible paths Γ_1^{is} for each node i.*

Proof: the finite termination directly follows from Proposition 3.5. Upon the termination, all acyclic paths between any node $i \neq s$ and s should have been examined since the procedure essentially performs a breadth-first search. Through FSD-CHECK, only acyclic paths that are not dominated will be kept in Γ_1^{is} at termination. Therefore the retained paths form a final Γ_1^{is}. □

It is clear that the complexity of FSD-LC depends on the size of Γ_1^{is}. In theory $|\Gamma_1^{is}|$ is only bounded by $|K^{is}|$, which grows exponentially with the number of nodes n (roughly n^{n-1} in the worse case). Thus, no algorithm of polynomial complexity exists for the RASP problem. We note that the RASP problem belongs to a class of multi-criteria shortest path problems, which are known to be intractable (e.g. Hansen 1979, Henig 1985, Miller-Hooks & Mahmassani 2000).

Proposition 3.6 *Algorithm FSD-LC runs in a non-polynomial time* $O(mn^{2n-1}L + mn^nL^2)$.

Proof: in the worst case, the algorithm may have to examine all possible paths for any O-D pair *is*. There are roughly $n \times n^{n-1} = n^{n-1}$ paths in total. For each path, all links may be scanned. Therefore Step 2 of the algorithm may be executed mn^n time. In step 2, $O(n)$ operations are required to check the acyclicity and $O(L^2)$ operations are required to calculate convolution integral. In FSD-CHECK, $O(L)$ and $O(n^{n-1}L)$ operations are consumed in Step 1 and 2, respectively. Thus the complexity is in the order of $O(mn^n(n + L^2 + L + n^{n-1}L))$, which yields the above result ignoring the $n + L$ portion in the parenthesis. □

In practice, we expect that $|K^{is}|$ is much smaller than n^{n-1}, in particular for sparse networks commonly seen in transportation applications. This has been noticed by a number of authors in numerical experiments (see Brumbaugh-Smith & Shier 1989, Miller-Hooks & Mahmassani 2000). Using the result of Henig (1985), one can show that the expected number of FSD-admissible paths is bounded by $\sum_{k=1}^{|K^{is}|} \frac{1}{k} \simeq \log(|K^{is}|)$ when $L = 1$ (i.e., two discrete time budgets). For $L > 1$, however, it is more difficult to establish such a theoretical bound. Through numerical examples, Wu & Nie (2009) showed that the relation between $|\Gamma_1^{is}|$ and n can be fitted using a quadratic function.

3.3.3 Discretize by probability

The discrete scheme proposed in the last section (Equation 3.8) requires $O(L^2)$ steps to calculate $u_k^{rs}(b)$. The size of L depends on both total travel time budget T and resolution ϕ. Though ϕ can be set independent of network, T is not. Ideally, T should equal the longest possible time to arrive at the destination from any origin. Essentially, this allows all trips to be completed with a probability up to 1.0. For example, if the longest possible travel time is 6 hours and $\phi = 5$ minutes, then $L = 6 \times 12 = 72$. Unfortunately, obtaining a good estimate of the maximum possible trip time itself is a hard problem for which no polynomial algorithms seem to exist (see Miller-Hooks & Mahmassani (1998) for a discussion of the least possible time path problem). To bypass this difficulty, one may simply set T to be a very large number. However, this brute-force treatment will raise computational issues as the complexity of the discrete algorithm highly depends on L. In a nutshell, the b-discrete method is not desirable because it leads to problem-specific complexity.

In the following, an alternative discretization method is proposed to overcome the shortcoming of b-discrete. Instead of discretizing the analysis period $[0, T]$, the new method considers a set of discrete points in the space of cumulative probability, namely $\alpha = \epsilon, 2\epsilon, \cdots, 1.0$, where $L\epsilon = 1.0$. We shall call this method α-discrete in the following. Corresponding to the α discrete points, a sequence of discrete travel times b_t^{ij} are generated for each link ij such that $0 = b_0^{ij} < b_1^{ij} < \cdots < b_t^{ij} < \cdots < b_L^{ij}$ and

$$b_t^{ij} = F_{ij}^{-1}(t\epsilon), \quad t = 1, \cdots, L \tag{3.10}$$

where $F_{ij}^{-1}(\cdot)$ is the inverse CDF of c_{ij}. Equation (3.10) implies

$$\int_{b_{t-1}^{ij}}^{b_t^{ij}} p_{ij}(w)dw = \epsilon, \quad t = 1, \cdots, L, \tag{3.11}$$

and with the Mean Value Theorem, we can always find $\hat{b}_t^{ij}$ for each interval $[b_{t-1}^{ij}, b_t^{ij}]$ such that

$$p_{ij}(\hat{b}_t^{ij})(b_t^{ij} - b_{t-1}^{ij}) = \epsilon \tag{3.12}$$

Thus, the probability mass function (PMF) in this discrete scheme is given by

$$P_{ij}(\hat{b}_t^{ij}) = \epsilon, \quad t = 1, \cdots, L \tag{3.13}$$

Accordingly, the distribution of path travel time π_k^{is} is represented by v_k^{is} instead of u_k^{is}. Given v_k^{js} and F_{ij}^{-1}, v_k^{is} can be approximately calculated using the following alternative convolution integral (ACI) procedure.

Procedure ACI

Step 0 set $\eta = 0$. For $t_1 = 1, \cdots L$; for $t_2 = 1, \cdots L$: set $\eta = \eta + 1, z_\eta = v_k^{js}(t_1\epsilon) + \hat{b}_{t_2}^{ij}$.

Step 1 Sort z_η in an ascending order.

Step 2 Construct the inverse CDF using $v_k^{is}(t\epsilon) = z_{tL},\ t = 1, \cdots L$.

In step 0, L^2 possible realizations of travel times are enumerated and stored in z_η. In Step 1, sorting a vector of length L^2 requires $O(L^2 \log L)$ steps if a binary tree is implemented. The last step consumes $O(L)$ steps. Thus, the complexity of procedure is dominated by the second step, which is higher than that of the discrete convolution (3.8) by a factor of $\log(L)$.

Note that L in the α-discrete method does not depend on T. Thus, the tradeoff between accuracy and computational cost can be easily controlled by selecting an ϵ, without considering network size and other problem-specific parameters that may impact T. Consequently, although α-discrete is more time-consuming than b-discrete for the same L, the extra computational overhead could be offset as α-discrete may lead to a smaller L.

Wu & Nie (2009) showed that the α-discrete method is much more efficient for some large network. For example it takes more than 2 hours to complete a single run using the b-discrete method for a 50×50 grid network (2500 nodes, 9,800 links), while it only takes less 1000 seconds using α-discrete method for the same network. Wu & Nie (2009) also showed that the α-discrete method is a competitive alternative to the b-discrete method. It not only make it possible to apply algorithm FSD-LC to large networks, but also provides reasonable approximation with comparable computational expense. The b-discrete method uses the uniformly spaced discrete points that are not effective in representing heterogeneous probability mass concentration. Especially they frequently overly represent the flat portions of a CDF. For example, numerical tests show that much more SD-admissible paths are solved using the b-discrete method. Most of them are non-dominated either in the interval $[0.99, 1]$ or $[0.01]$. Because no support points in these two intervals when using the α-discrete method (resolution of probability is 0.01), these non-dominated paths are missed when using the α-discrete method. However, these missed non-dominated paths are almost useless in practice: few travelers would be sensitive to an improvement of less than 1% on-time arrival reliability. More importantly, less non-dominated paths (i.e., smaller $|\Gamma_1^{rs}|$) help Algorithm FSD-LC run much faster.

3.3.4 A hybrid discretization approach

The b-discrete and α-discrete methods both employ the uniformly spaced discrete points. In the b-discrete method, the travel time budget is uniformly discretized; while in the α-discrete method, the probability mass is uniformly discretized. However, the uniformly spaced discrete points are not effective in representing heterogenous probability mass concentration. They frequently overly represent the flat portions, while at times fail to capture the hot spots where rapid changes take place. In light of the above limitation, this study adopts a hybrid approach to allow both variable length of discrete intervals and different probability mass in each interval. The hybrid approach starts from a set of L uniform intervals (whose length may vary from one random variable to another), and computes the probability mass functions with Equation 3.7. However, a *consolidation* procedure is incorporated to merge consecutive intervals together such that no more than one interval has a probability mass smaller than $1/L$. "Merging" two consecutive discrete intervals means removing the boundary between them and assigning the sum of their probability masses to the new interval. The consolidation produces a set of effective support intervals (ESI), whose size is often much smaller than L according to our experience. Consequently, the hybrid approach brings about significant computational benefits, without compromising the accuracy and stability of numerical convolution.

We now show how the convolution can be performed using the hybrid discretization scheme. Consider two random variables X and Y, whose domain is represented (after discretization and consolidation), respectively, by the following sets of break points: $[x_0, x_1, ..., x_{L(X)}]$, $[y_0, y_1, ..., y_{L(Y)}]$, where $L(X)$ and $L(Y)$ are numbers of ESI. Accordingly, the discrete support points are defined as

$$S^X = [\frac{x_0 + x_1}{2}, \cdots, \frac{x_{L(X)-1} + x_{L(X)}}{2}], S^Y = [\frac{y_0 + y_1}{2}, \cdots, \frac{y_{L(Y)-1} + y_{L(Y)}}{2}],$$

and the probability mass functions are

$$P^X = [P_1^X, \cdots, P_{L(X)}^X], P^Y = [P_1^Y, \cdots, P_{L(Y)}^Y].$$

The following method can be used to compute P^Z for $Z = X \oplus Y$, where $\oplus$ denotes convolution integral. Denote $P0^Z$ be the probability mass function before consolidation.

Convolution based on the hybrid discretization approach

Step 0 Set $z_{\min} = x_0 + y_0, z_{\max} = x_{L(X)} + Y_{L(Y)}$. Divide $[z_{\min}, z_{\max}]$ into L intervals of uniform length, and compute $\phi = (z_{\max} - z_{\min})/L$. Intialize $P0_l^Z = 0, \forall l = 1, \cdots, L$.

```
Step 1 for $i = 1, 2, \cdots, L(X)$,
    for $j = 1, 2, \cdots, L(Y)$
        Calculate $ts = S_i^X + S_j^Y$ and $tp = P_i^X \cdot P_j^Y$. Define $l = \left[\frac{ts - z_{\min}}{\phi}\right]_-$, and set $P0_l^Z = P0_l^Z + tp$
    end for
  end for
```

Step 2 Consolidate $P0^Z$ to get effective support intervals and the associated probability mass functions.

The procedure of consolidation is given as follows.

Procedure: Consolidation

Inputs: $P0^Z$ from convolution and $z = \{z_0, z_1, \cdots, z_L\}$, where elements are evenly distributed from $z_{\min}$ to $z_{\max}$ with the step size $[z_{\max} - z_{\min}]/L$, and the desired probability resolution ϵ.

Outputs: S^Z contains the effective support intervals and P^Z contains the probability mass corresponding to each element in S^Z.

Step 0 Calculate two temporary vectors $\phi = \{P0_1^Z - P0_0^Z, P0_2^Z - P0_1^Z, \cdots, P0_L^Z - P0_{L-1}^Z\}$ and $\psi = \{\frac{z_0 + z_1}{2}, \frac{z_1 + z_2}{2}, \cdots, \frac{z_{L-1} + z_L}{2}\}$, respectively. Travel time realization ψ_t corresponds to probability ϕ_t, $t = 0, 1, \cdots, L - 1$.

Step 1 Begin with $t_0 = 0, \kappa = 0$. Let $P^Z_\kappa = \sum_{t=t_0}^{\eta} \phi_t$, where $\eta < L$, $\sum_{t=t_0}^{\eta-1} \phi_t < \epsilon$ and $\sum_{t=t_0}^{\eta} \phi_t \geq \epsilon$, and $S^Z_\kappa = \sum_{t=t_0}^{\eta} \psi_t \cdot \phi_t$. Then $t_0 = \eta$, $\kappa = \kappa + 1$

Step 2 Repeat Step 2 until $\eta = L$. Let $L_{ij} = \kappa$.

Finally, we note that the determination of FSD-admissibility relies on comparing CDFs. In the hybrid discretization scheme, the CDF of any consolidated distribution can be evaluated at any feasible point using linear interpolation. Thus, FSD-admissibility can be examined at any desired resolution and is not restricted by the discrete points used to represent distributions. The hybrid discretization scheme is employed in our case study.

Chapter 4

Implementation and Graphic User Interface

The algorithm described in the last chapter is implemented on the top of Toolkit of Network Modeling (TNM), a C++ class library for solving various transportation network problems. Thanks to the object-oriented design adopted in TNM, we are able to easily fit new code into the existing class hierarchy, which has greatly reduced the programming efforts.

4.1 Class Hierarchy

This section provides an overview of class hierarchy, with a focus on new classes introduced in this research. A detailed class implementation can be found in Appendix A.

Originally, TNM defines four major network classes:

- `TNM_SNET`: for static applications such as traffic assignment.
- `TNM_DNET`: for macroscopic dynamic applications such as dynamic network loading and dynamic traffic assignment.
- `TNM_MNET`: for microscopic dynamic applications, such as studying vehicles' lane-changing behavior.
- `TNM_ProbeNet`: reserved for stochastic applications.

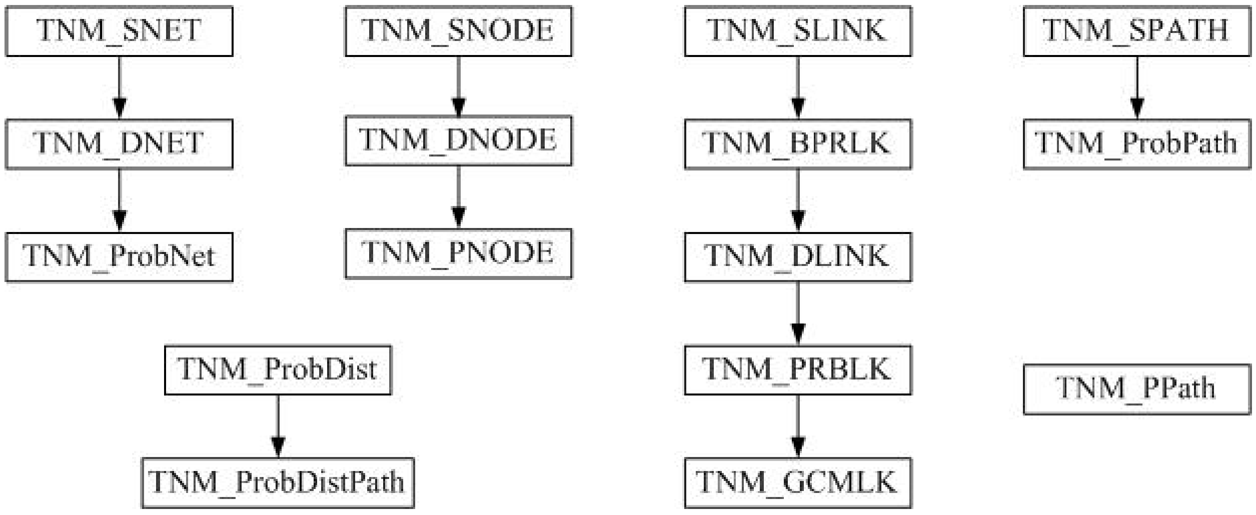


Figure 4.1: A class hierarchy tree

All the four network classes are derived from TNM_SNET, whose major data members include, among others, lists of *node* and *link* which together represent the topology of the network. Each type of network is usually associated with link and node objects of corresponding types. For example, Class TNM_SLINK is the basic link type for TNM_SNET, while Class TNM_DLINK is used by TNM_DNET. For a detailed description of these classes, the reader is referred to (Nie 2006).

As shown in Figure 4.1, TNM_ProbeNet is derived from TNM_DNET. While this class exists in the original TNM, it has been substantially expanded in this research to support reliable routing. The important new functions include generating non-dominant paths, I-O functions for travel time distributions, as well as build functions for new network format.

In addition, new node and link classes are introduced. TNM_PNODE, derived from TNM_DNODE, is used to construct, store, manipulate and compare paths. Most important operations defined in Algorithm FSD-LC are actually implemented in this class. TNM_PRBLK and TNM_GCMLINK are derived classes of TNM_DLINK. No important methods are defined in these classes. Instead, they are primarily holders of new data members, such as detector information and road names.

TNM_ProbDist is a new class intended to handle discrete probability distribution. In addition to regular functionality, the class provides an efficient implementation of convolution integral using the hybrid discretization scheme introduced in Chapter 3. It also allows one to easily compare one distribution with another using stochastic dominance. `TNM_ProbDist` is further wrapped by class `TNM_PPATH`, which implements connection among paths stored at adjacent nodes. Through this connection, one can construct a path from its pointer at any given node. Finally, to be consistent with the existing hierarchy, a new class `TNM_ProbPATH` is derived from the existing path class `TNM_SPATH` and wraps `TNM_PPATH`.

4.2 Graphic user interface

A graphic user interface (`GUI`), named **CTR** (Chicago Travel Reliability), was developed using `MFC` and `MYSQL`. CTR offers a range of useful visualization functions intended to provide an integrated environment to: 1) visualize and analyze traffic data, 2) construct and display travel reliability measures for the Chicago network, and 3) provide reliable route guidance and compare it with conventional routing algorithms.

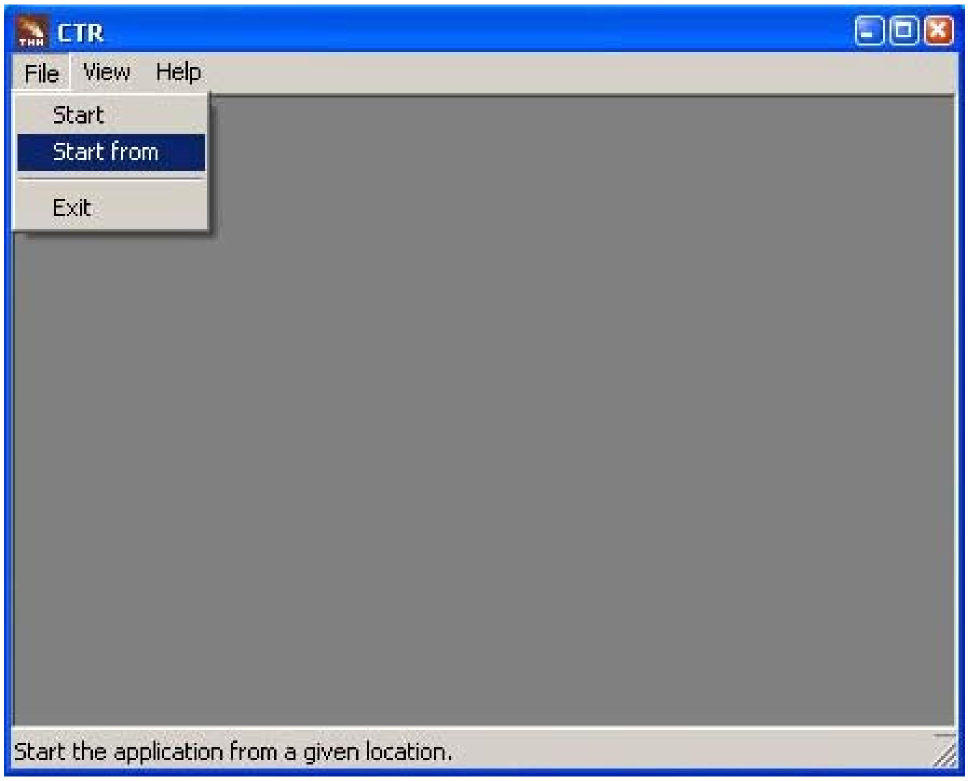
Figures 4.2(a) and 4.2(b) show the CTR window before and after loading the Chicago network (select `file` → `start` to load network from the default folder, or `file` → `start from` if you would like to load the Chicago network from a different location). After the network is loaded, right click on the map to zoom in/out or pan the current view, as shown in Figure 4.2(b). The map functions can also be accessed through toolbar or the dropdown menu. Double click on the map (or use `Map`→`Setting`) to modify map settings.

Figures 4.3(a)-4.3(c) show four dropdown menus of CTR, which are briefly described in the following.

- Control

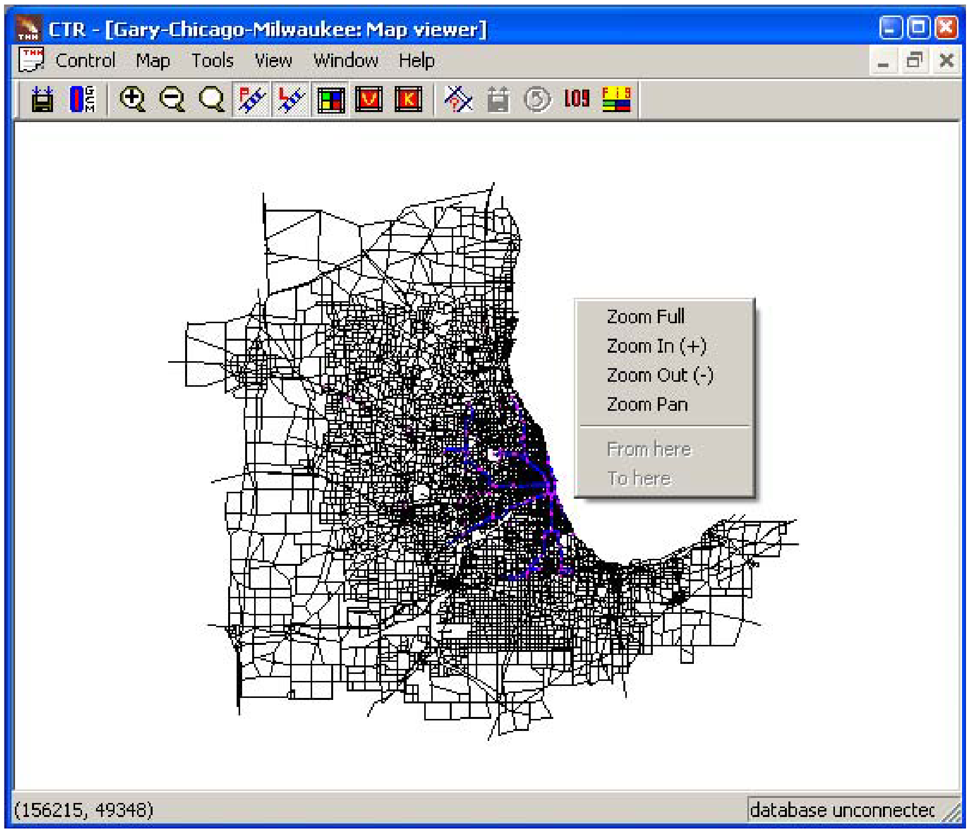
  Save Overwrite the loaded network file with current editing. Normally the user is not expected to use this function.

  Save Map View Save the portion of the network appeared in the current view window as an independent network file. The user can load the saved network file later. This function helps create smaller instances of the routing problem.



Start the application from a given location.

(a) CTR window, before launching the network file

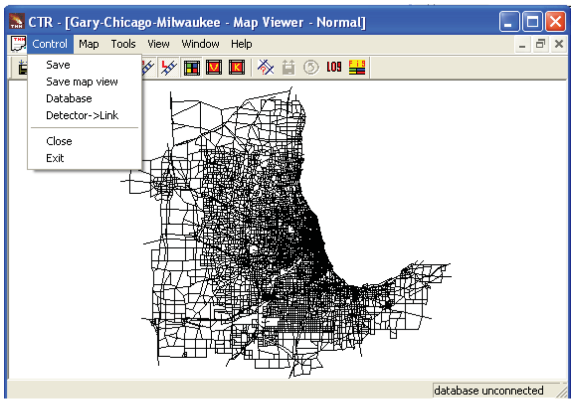


(156215, 49348)

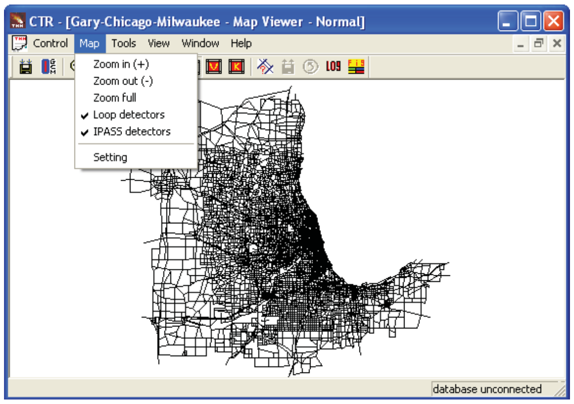
database unconnectec

(b) CTR window, after launching the GCM network

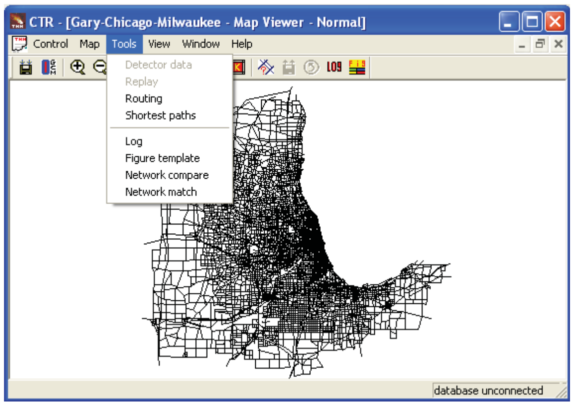
Figure 4.2: CTR Interface


CTR - [Gary-Chicago-Milwaukee - Map Viewer - Normal]
Control Map Tools View Window Help
Save
Save map view
Database
Detector->Link
Close
Exit
database unconnected

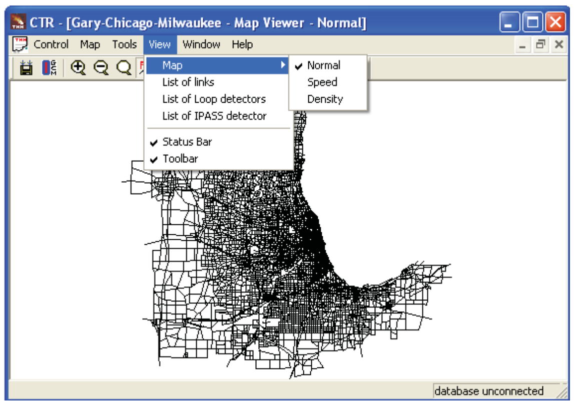
(a) ”Control” menu



(b) ”Map menu



(c) ”Tool” menu



(d) ”View” menu

Figure 4.3: CTR main dropdown menus

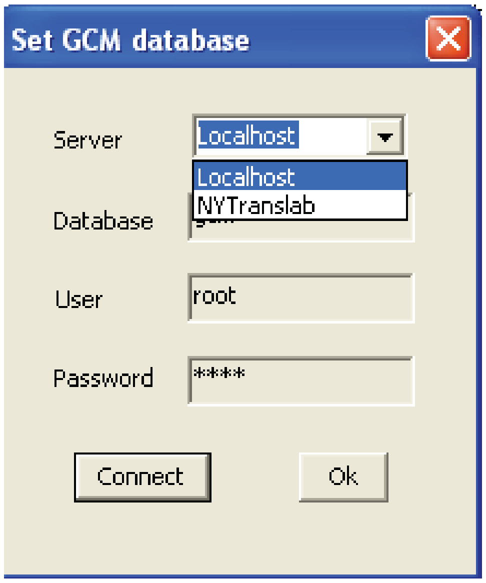


Figure 4.4: Database connection dialog

Database Connect to the mysql database. Most functions require to connect to the database first, particularly those related to traffic data inquiries. Figure 4.4 shows the dialog window for database connection. To connect to the database, select NUTranlab as server and click ”Connect”.

Detector→link Create a mapping between detectors and links. Normally the user does not need this function.

- Map

Zoom in Zoom in the map (increase size).

Zoom out Zoom out the map (decrease size).

Zoom full Return to the full view of the network.

Loop detectors Show / hide loop detectors on the map.

IPASS detectors Show / hide IPASS detectors on the map.

- View

Map Switch map view pattern. The map has three patterns: normal view; speed view, which colors each link with speed data read from database (useful when traffic data is displayed dynamically), density view, which colors each link with occupancy data read from database (useful when traffic data is displayed dynamically).

List of links Open a link view (see Figure 4.5(a)). Link view and map view are linked together inherently. That is, when you click a link on the link view, the map view will automatically zoom into the link and highlight it. This makes it easier to locate a link on a map according to its ID (and other properties) or vice versa.

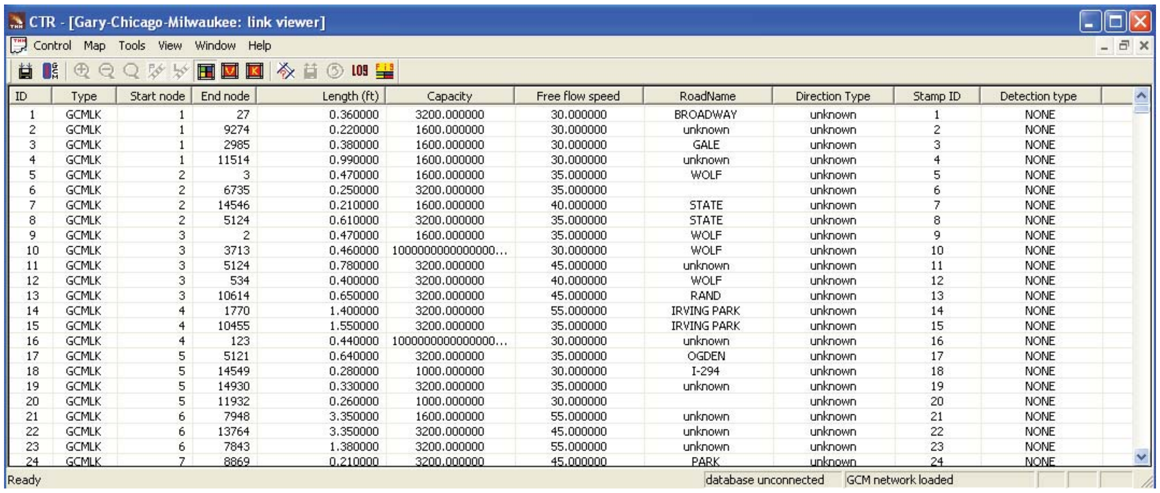
List of loop detectors Open a loop detector view (see Figure 4.5(b)). Loop detector view is linked to map view in a similar way as the link view. Double click on a row in the loop detector view will pop out a data inquire window as shown in Figure 4.6 (the mysql database has to be connected first). In this dialog, the user can plot the distribution for the selected criteria (time of day, season etc) and traffic quantity (speed, occupancy etc). The dialog can also plot the relationship between speed, occupancy and flow rates (click on `relation`).

List of IPASS detectors Open an IPASS view (see Figure 4.5(c)). IPASS view is also linked to map view and double click on a row of the view will bring the user to the same data inquiry dialog as for loop detector view. The difference is that the user cannot select traffic quantity (since travel time is the only possible option) and the plot type (only `distribution` plots can be made).

- Tools

Detector data Provides a data inquiry tool (see Figure 4.7). The user can plot the profiles of traffic quantities over time and compare them for different days and detectors.

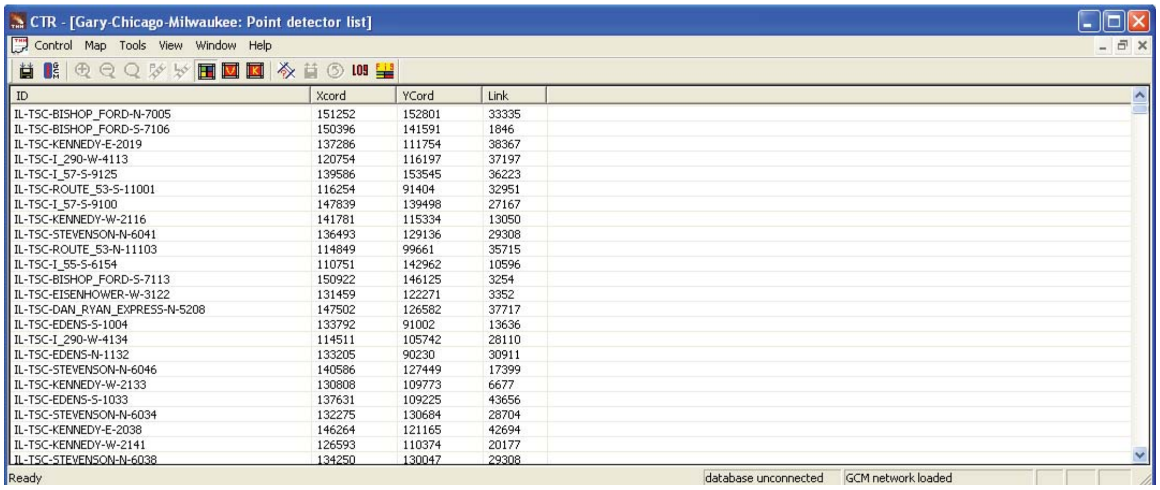
Replay Allow users to dynamically visualize traffic data on the map for a given day by coloring links. This function can be called only when an active map view exists in either `speed` or `density` pattern. Once the replay dialog is activated, first select a day and data source (from detector, IPASS or both), then click play to start the animation, see Figure 4.8). The user can drag the


CTR - [Gary-Chicago-Milwaukee: link viewer]
Control Map Tools View Window Help
LOG

ID	Type	Start node	End node	Length (ft)	Capacity	Free flow speed	RoadName	Direction Type	Stamp ID	Detection type
1	GCMLK	1	27	0.360000	3200.000000	30.000000	BROADWAY	unknown	1	NONE
2	GCMLK	1	9274	0.220000	1600.000000	30.000000	unknown	unknown	2	NONE
3	GCMLK	1	2985	0.380000	1600.000000	30.000000	GALE	unknown	3	NONE
4	GCMLK	1	11514	0.990000	1600.000000	30.000000	unknown	unknown	4	NONE
5	GCMLK	2	3	0.470000	1600.000000	35.000000	WOLF	unknown	5	NONE
6	GCMLK	2	6735	0.250000	3200.000000	35.000000		unknown	6	NONE
7	GCMLK	2	14546	0.210000	1600.000000	40.000000	STATE	unknown	7	NONE
8	GCMLK	2	5124	0.610000	3200.000000	35.000000	STATE	unknown	8	NONE
9	GCMLK	3	2	0.470000	1600.000000	35.000000	WOLF	unknown	9	NONE
10	GCMLK	3	3713	0.460000	1000000000000000...	30.000000	WOLF	unknown	10	NONE
11	GCMLK	3	5124	0.780000	3200.000000	45.000000	unknown	unknown	11	NONE
12	GCMLK	3	534	0.400000	3200.000000	40.000000	WOLF	unknown	12	NONE
13	GCMLK	3	10614	0.650000	3200.000000	45.000000	RAND	unknown	13	NONE
14	GCMLK	4	1770	1.400000	3200.000000	55.000000	IRVING PARK	unknown	14	NONE
15	GCMLK	4	10455	1.550000	3200.000000	35.000000	IRVING PARK	unknown	15	NONE
16	GCMLK	4	123	0.440000	1000000000000000...	30.000000	unknown	unknown	16	NONE
17	GCMLK	5	5121	0.640000	3200.000000	35.000000	OGDEN	unknown	17	NONE
18	GCMLK	5	14549	0.280000	1000.000000	30.000000	I-294	unknown	18	NONE
19	GCMLK	5	14930	0.330000	3200.000000	35.000000	unknown	unknown	19	NONE
20	GCMLK	5	11932	0.260000	1000.000000	30.000000		unknown	20	NONE
21	GCMLK	6	7948	3.350000	1600.000000	55.000000	unknown	unknown	21	NONE
22	GCMLK	6	13764	3.350000	3200.000000	45.000000	unknown	unknown	22	NONE
23	GCMLK	6	7843	1.380000	3200.000000	55.000000	unknown	unknown	23	NONE
24	GCMLK	7	8869	0.210000	3200.000000	45.000000	PARK	unknown	24	NONE

Ready | database unconnected | GCM network loaded

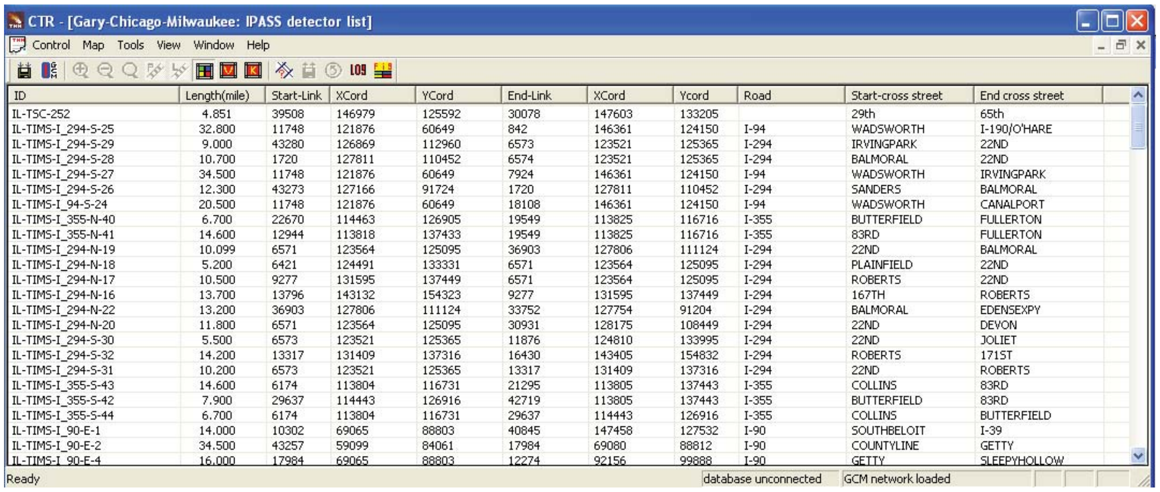
(a) Link View


CTR - [Gary-Chicago-Milwaukee: Point detector list]
Control Map Tools View Window Help
LOG

ID	Xcord	YCord	Link
IL-TSC-BISHOP_FORD-N-7005	151252	152801	33335
IL-TSC-BISHOP_FORD-S-7106	150396	141591	1846
IL-TSC-KENNEDY-E-2019	137286	111754	38367
IL-TSC-I_290-W-4113	120754	116197	37197
IL-TSC-I_57-S-9125	139586	153545	36223
IL-TSC-ROUTE_53-S-11001	116254	91404	32951
IL-TSC-I_57-S-9100	147839	139498	27167
IL-TSC-KENNEDY-W-2116	141781	115334	13050
IL-TSC-STEVENSON-N-6041	136493	129136	29308
IL-TSC-ROUTE_53-N-11103	114849	99661	35715
IL-TSC-I_55-S-6154	110751	142962	10596
IL-TSC-BISHOP_FORD-S-7113	150922	146125	3254
IL-TSC-EISENHOWER-W-3122	131459	122271	3352
IL-TSC-DAN_RYAN_EXPRESS-N-5208	147502	126582	37717
IL-TSC-EDENS-S-1004	133792	91002	13636
IL-TSC-I_290-W-4134	114511	105742	28110
IL-TSC-EDENS-N-1132	133205	90230	30911
IL-TSC-STEVENSON-N-6046	140586	127449	17399
IL-TSC-KENNEDY-W-2133	130808	109773	6677
IL-TSC-EDENS-S-1033	137631	109225	43656
IL-TSC-STEVENSON-N-6034	132275	130684	28704
IL-TSC-KENNEDY-E-2038	146264	121165	42694
IL-TSC-KENNEDY-W-2141	126593	110374	20177
IL-TSC-STEVENSON-N-6038	134250	130047	29308

Ready | database unconnected | GCM network loaded

(b) Loop detector view

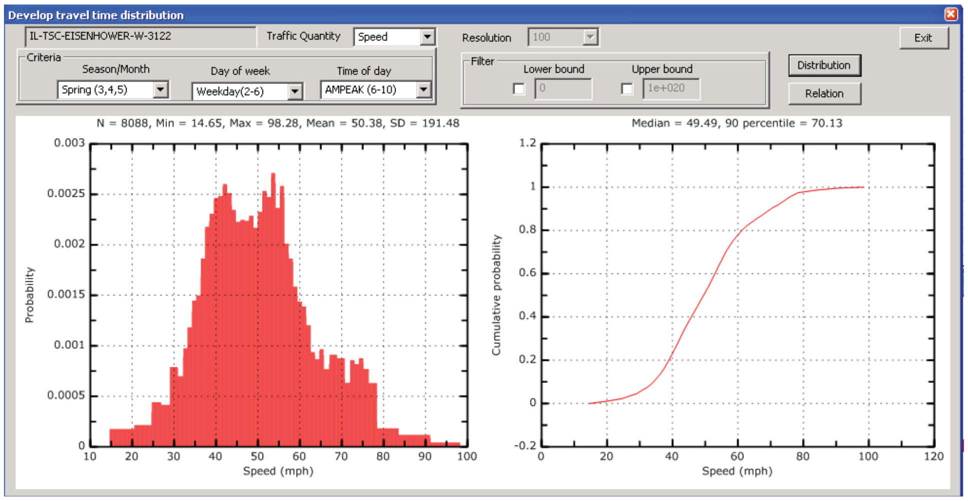

CTR - [Gary-Chicago-Milwaukee: IPASS detector list]
Control Map Tools View Window Help
LOG

ID	Length(mile)	Start-Link	XCord	YCord	End-Link	XCord	Ycord	Road	Start-cross street	End cross street
IL-TSC-252	4.851	39508	146979	125592	30078	147603	133205		29th	65th
IL-TIMS-I_294-S-25	32.800	11748	121876	60649	842	146361	124150	I-94	WADSWORTH	I-190/O'HARE
IL-TIMS-I_294-S-29	9.000	43280	126869	112960	6573	123521	125365	I-294	IRVINGPARK	22ND
IL-TIMS-I_294-S-28	10.700	1720	127811	110452	6574	123521	125365	I-294	BALMORAL	22ND
IL-TIMS-I_294-S-27	34.500	11748	121876	60649	7924	146361	124150	I-94	WADSWORTH	IRVINGPARK
IL-TIMS-I_294-S-26	12.300	43273	127166	91724	1720	127811	110452	I-294	SANDERS	BALMORAL
IL-TIMS-I_94-S-24	20.500	11748	121876	60649	18108	146361	124150	I-94	WADSWORTH	CANALPORT
IL-TIMS-I_355-N-40	6.700	22670	114463	126905	19549	113825	116716	I-355	BUTTERFIELD	FULLERTON
IL-TIMS-I_355-N-41	14.600	12944	113818	137433	19549	113825	116716	I-355	83RD	FULLERTON
IL-TIMS-I_294-N-19	10.099	6571	123564	125095	36903	127806	111124	I-294	22ND	BALMORAL
IL-TIMS-I_294-N-18	5.200	6421	124491	133331	6571	123564	125095	I-294	PLAINFIELD	22ND
IL-TIMS-I_294-N-17	10.500	9277	131595	137449	6571	123564	125095	I-294	ROBERTS	22ND
IL-TIMS-I_294-N-16	13.700	13796	143132	154323	9277	131595	137449	I-294	167TH	ROBERTS
IL-TIMS-I_294-N-22	13.200	36903	127806	111124	33752	127754	91204	I-294	BALMORAL	EDENSEXPY
IL-TIMS-I_294-N-20	11.800	6571	123564	125095	30931	128175	108449	I-294	22ND	DEVON
IL-TIMS-I_294-S-30	5.500	6573	123521	125365	11876	124810	133995	I-294	22ND	JOLIET
IL-TIMS-I_294-S-32	14.200	13317	131409	137316	16430	143405	154832	I-294	ROBERTS	171ST
IL-TIMS-I_294-S-31	10.200	6573	123521	125365	13317	131409	137316	I-294	22ND	ROBERTS
IL-TIMS-I_355-S-43	14.600	6174	113804	116731	21295	113805	137443	I-355	COLLINS	83RD
IL-TIMS-I_355-S-42	7.900	29637	114443	126916	42719	113805	137443	I-355	BUTTERFIELD	83RD
IL-TIMS-I_355-S-44	6.700	6174	113804	116731	29637	114443	126916	I-355	COLLINS	BUTTERFIELD
IL-TIMS-I_90-E-1	14.000	10302	69065	88803	40845	147458	127532	I-90	SOUTHBELOIT	I-39
IL-TIMS-I_90-E-2	34.500	43257	59099	84061	17984	69080	88812	I-90	COUNTYLINE	GETTY
IL-TIMS-I_90-E-4	16.000	17984	69065	88803	12274	92156	99888	I-90	GETTY	SLEEPYHOLLOW

Ready | database unconnected | GCM network loaded

(c) IPASS view

Figure 4.5: Different views available in CTR


Develop travel time distribution

IL-TSC-EISENHOWER-W-3122 — Traffic Quantity: Speed — Resolution: 100

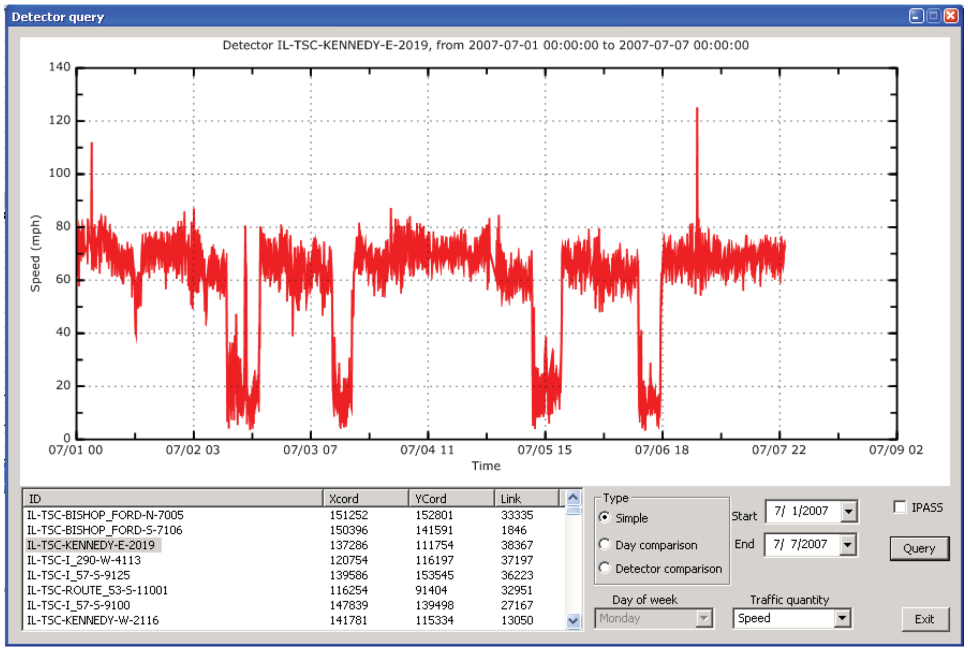
Criteria — Season/Month: Spring (3,4,5) — Day of week: Weekday(2-6) — Time of day: AMPEAK (6-10)

Filter — Lower bound: 0 — Upper bound: 1e+020

N = 8088, Min = 14.65, Max = 98.28, Mean = 50.38, SD = 191.48

Median = 49.49, 90 percentile = 70.13

Figure 4.6: Distributions of traffic quantities


Detector query

Detector IL-TSC-KENNEDY-E-2019, from 2007-07-01 00:00:00 to 2007-07-07 00:00:00

ID	Xcord	YCord	Link
IL-TSC-BISHOP_FORD-N-7005	151252	152801	33335
IL-TSC-BISHOP_FORD-S-7106	150396	141591	1846
IL-TSC-KENNEDY-E-2019	137286	111754	38367
IL-TSC-I_290-W-4113	120754	116197	37197
IL-TSC-I_57-S-9125	139586	153545	36223
IL-TSC-ROUTE_53-S-11001	116254	91404	32951
IL-TSC-I_57-S-9100	147839	139498	27167
IL-TSC-KENNEDY-W-2116	141781	115334	13050

Type: [✓ Simple] Day comparison Detector comparison

Start: 7/ 1/2007 — End: 7/ 7/2007 — IPASS

Day of week: Monday — Traffic quantity: Speed

Figure 4.7: Traffic data inquiry

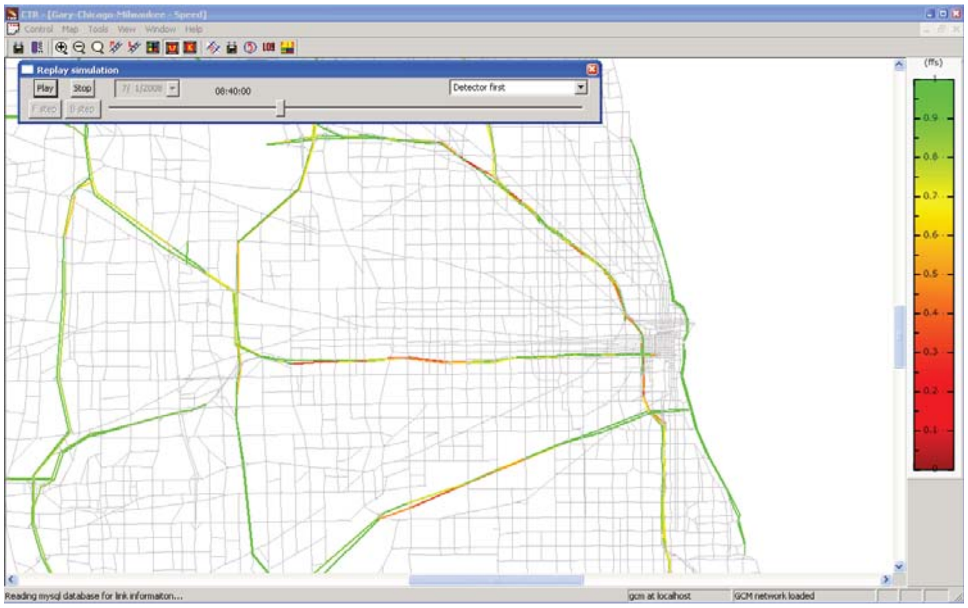


Figure 4.8: Dynamic display of traffic quantities

button on the slide bar to any time of the day, or use `F step` (forward) and `B step` (backward) to manually control time.

Shortest path Implements a conventional shortest path algorithm.

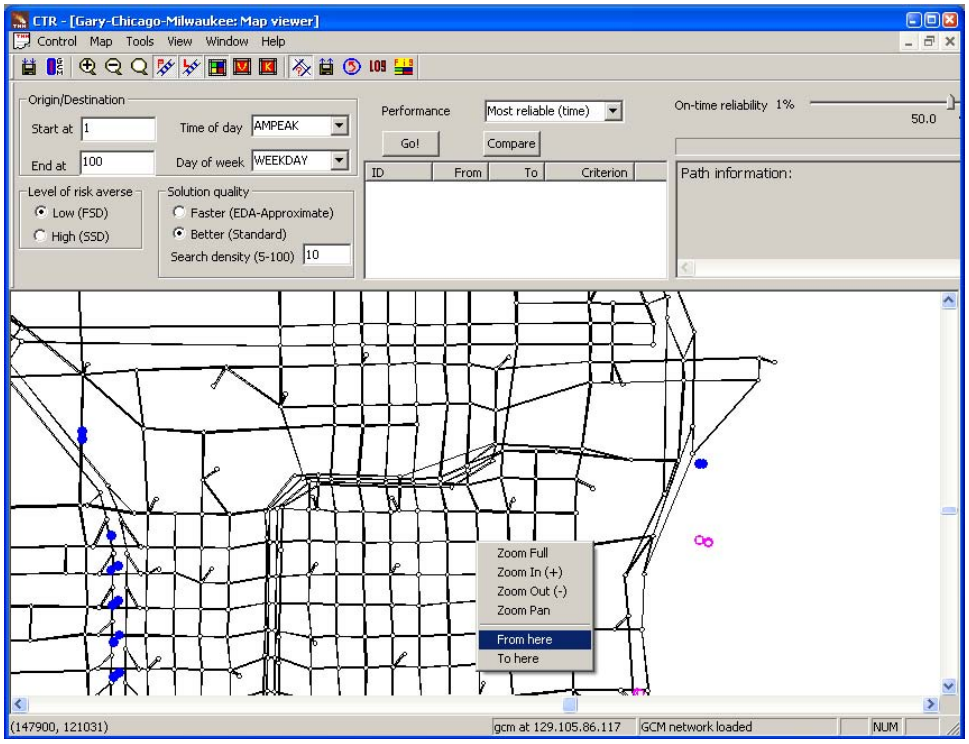
Log Display running information in a log window, such as warning or error messages which are otherwise invisible to the user.

Network compare Compare two networks' topology, not intended for regular users.

Network match Match two networks's topology, not intended for regular users.

Figure template A PGL graph editor. All plots produced by CTR are in PGL format and can be saved as a PGL file. The saved PGL files can be loaded and edited using this function and exported to other formats such as jpeg, eps, etc.

Routing The core component in CTR that integrates functions related to reliable routing. Note that the connection with the mysql database has to be built first in order to properly run this function. The first step is to select the origin and destination. This can be done directly in the panel by inputting the node IDs. Alternatively, right click on the map, and select "From here" or "To here" for origin or destination respectively (see Figure 4.9). The user can also specify the time of day (AM Peak, PM Peak, etc) and day of week (weekday, weekend), which will determine the corresponding travel time distribution employed in the reliable routing. The user can also select several algorithmic parameters, such as FSD vs. SSD (second-order stochastic dominance), higher solution quality vs. better computational performance and search density. The recommended choice is FSD + Better + 10 (search density). Other choices should be considered only when routing takes too long to accomplish. Also note that a larger search density may significantly increase computational time and increase the number of admissible paths. After all parameters are properly set, click Go to solve the RASP problem. Admissible paths for the selected O-D pair will appear in the window under the ”Go” button after the routing operation is terminated (see Figure 4.10). The user can select one or more admissible paths from the path and compare them (click compare button). The compare function will generate a window that plots CDFs of travel times on the selected paths (see Figure 4.11). The user can also move the slide bar to adjust the current on-time reliability. Whenever the on-time reliability is selected on the bar, the corresponding minimum travel time budget will be displayed in the text field window, and the optimal path will be highlighted in both the path view window and on the map.


CTR - [Gary-Chicago-Milwaukee: Map viewer]
Control Map Tools View Window Help
Go!
Compare
Zoom Full
Zoom In (+)
Zoom Out (-)
Zoom Pan
From here
To here
NUM

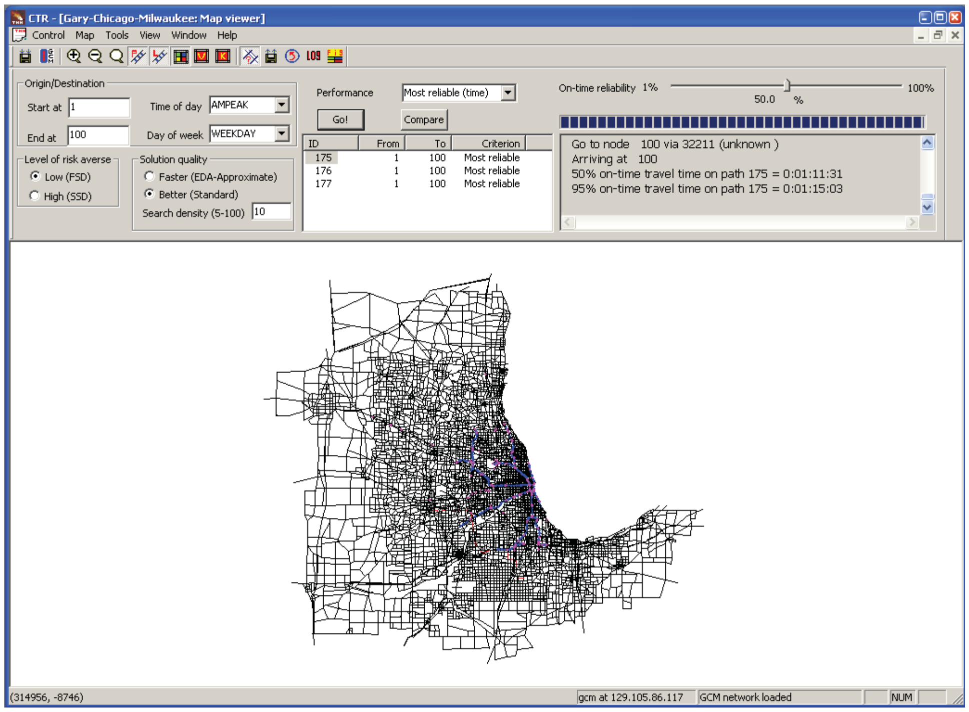
Origin/Destination
Start at 1
End at 100
Time of day AMPEAK
Day of week WEEKDAY
Level of risk averse
Low (FSD) [✓]
High (SSD)
Solution quality
Faster (EDA-Approximate)
Better (Standard) [✓]
Search density (5-100) 10
Performance Most reliable (time)
On-time reliability 1% 50.0

ID	From	To	Criterion

Path information:

(147900, 121031) gcm at 129.105.86.117 GCM network loaded

Figure 4.9: CTR routing panel


CTR - [Gary-Chicago-Milwaukee: Map viewer]
Control Map Tools View Window Help
Go!
Compare
NUM

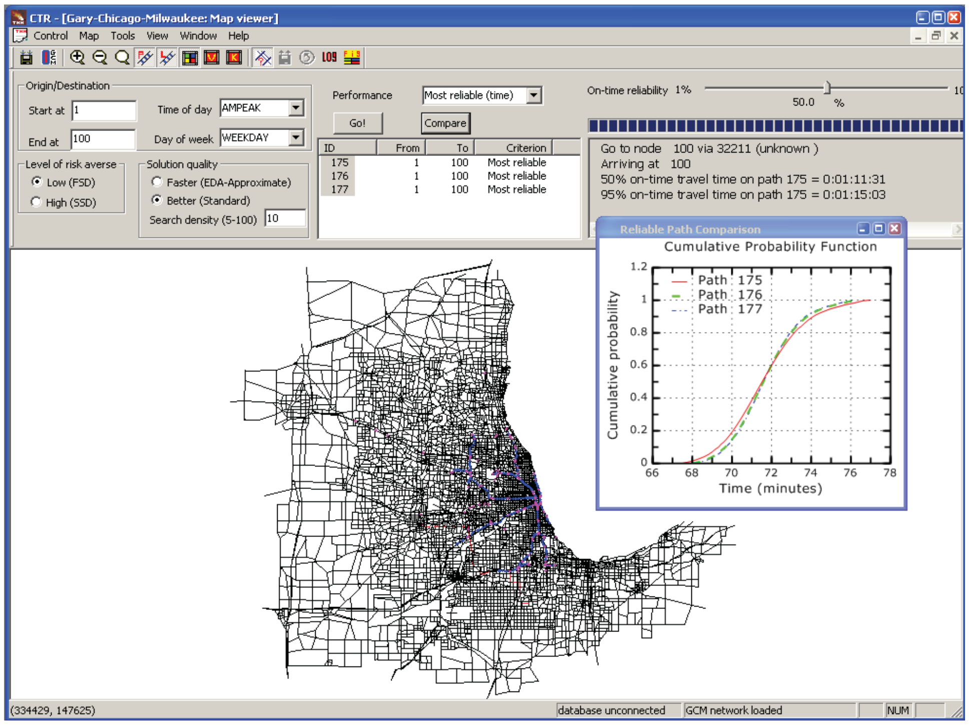
Origin/Destination
Start at 1
End at 100
Time of day AMPEAK
Day of week WEEKDAY
Level of risk averse
Low (FSD) [✓]
High (SSD)
Solution quality
Faster (EDA-Approximate)
Better (Standard) [✓]
Search density (5-100) 10
Performance Most reliable (time)
On-time reliability 1% 50.0 % 100%

ID	From	To	Criterion
175	1	100	Most reliable
176	1	100	Most reliable
177	1	100	Most reliable

```
Go to node   100 via 32211 (unknown )
Arriving at   100
50% on-time travel time on path 175 = 0:01:11:31
95% on-time travel time on path 175 = 0:01:15:03
```

(314956, -8746) gcm at 129.105.86.117 GCM network loaded

Figure 4.10: Display of Routing results



Origin/Destination
Start at 1
End at 100
Time of day AMPEAK
Day of week WEEKDAY
Level of risk averse: [✓ Low (FSD)] High (SSD)
Solution quality: Faster (EDA-Approximate) [✓ Better (Standard)]
Search density (5-100) 10
Performance Most reliable (time)
Go! Compare
On-time reliability 1% 50.0 %

ID	From	To	Criterion
175	1	100	Most reliable
176	1	100	Most reliable
177	1	100	Most reliable

```
Go to node   100 via 32211 (unknown )
Arriving at  100
50% on-time travel time on path 175 = 0:01:11:31
95% on-time travel time on path 175 = 0:01:15:03
```

(334429, 147625) database unconnected GCM network loaded NUM

Figure 4.11: Compare routing results in CTR

Chapter 5

The Case Study and Data

The Chicago region is selected for a case study to test reliable routing algorithms because of two reasons. First, as the third largest metropolitan area in the US, the city of Chicago and its neighboring suburban areas are subject to significant congestion. According to the latest mobility report Schrank & Lomax (2007), an average commuter in Chicago area wasted 46 hours due to traffic congestion in 2007. Perhaps more important, the travel times in the Chicago area seem more unreliable than any other major metropolitan areas in the US. The report indicates that a traveler in Chicago area has to budget 2.07 times of the free flow travel time for an important trip (which requires 95% probability of on-time arrival), the highest index in the country. Second, Chicago has archived a rich set of traffic data in both public and private sectors. In particular, the GCM (Gary-Chicago-Milwaukee corridor) traveler information system (www.gcmtravel.com) broadcasts and archives real-time traffic data collected from loop detectors, toll transponders and other devices operated by departments of transportation in Illinois, Wisconsin and Indiana. Notably, this database provides (point-to-point) real-time travel time for most toll roads in the region since 2004.

This research uses GCM database as the the primary source for traffic data on freeways and toll roads. The GCM data come from two main sources: loop detectors and electronic toll transponders (IPASS), which cover freeways and toll roads respectively. Figure 5.1 shows a topology of the Chicago network used in the case study, as well as the location of loop detectors and IPASS toll booths. The network data are from the latest travel planning model prepared by the Chicago Metropolitan Agency for Planning

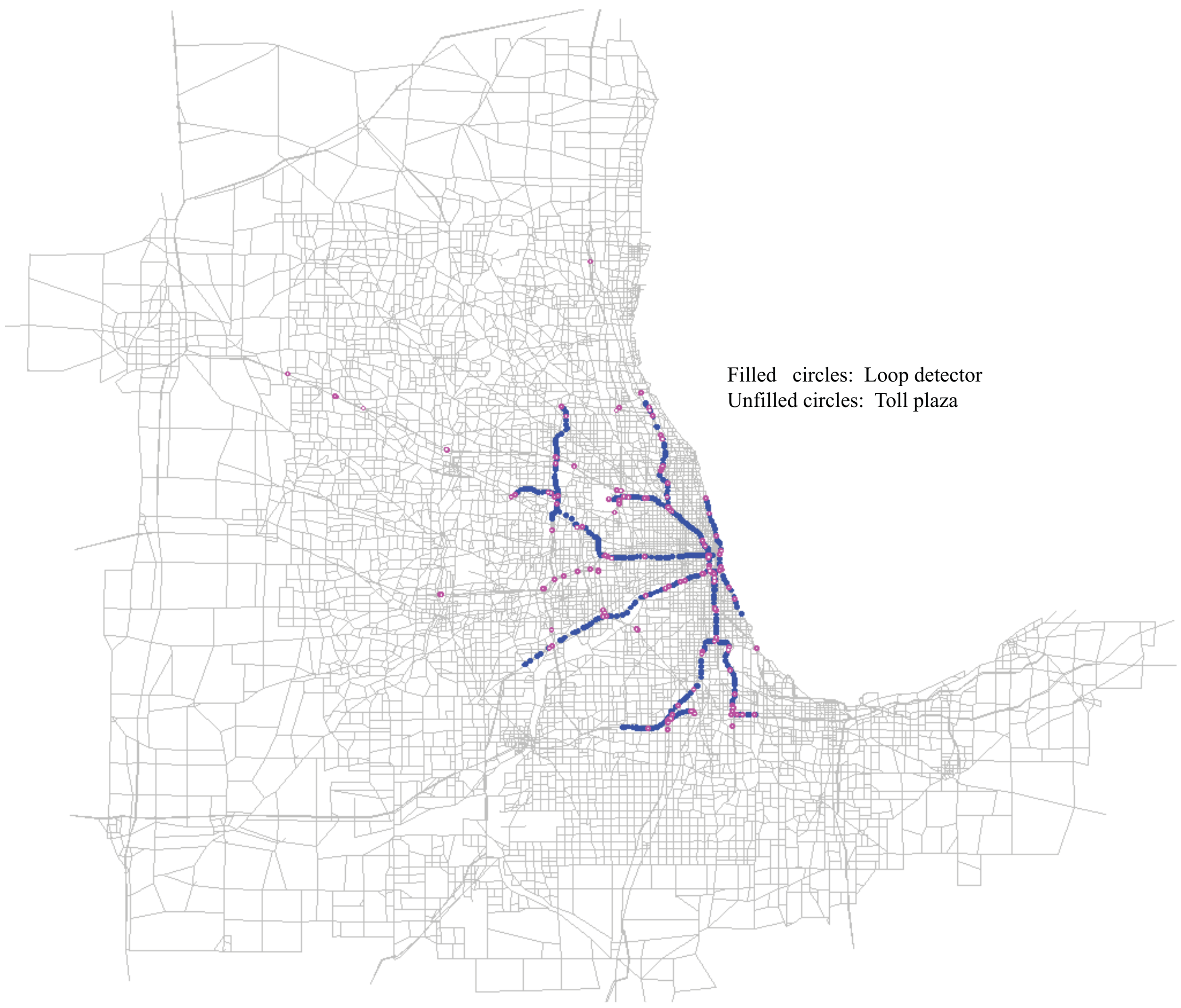


Figure 5.1: The Chicago network from Chicago Metropolitan Agency for Planning

(CMAP). We note that the GCM system provides its own network data. However, the GCM network has two major drawbacks: it does not provide basic link properties such as speed limit and capacity; and its links are short in order to depict fine geometric features, which leads to unnecessarily large network representation so far as routing is concerned. The CMAP network is adopted in light of these shortcomings.

As revealed in Figure 5.1, a major problem is the lack of data on arterial and local streets, which constitute the majority of links in the network. Recognizing that this is a universal problem that has not yet been overcome in the current paradigm of data collection, we estimate travel time distributions on these streets using results from a travel planning model.

5.1 Data for freeway and toll roads

In the GCM database, loop detectors (primarily operated by Illinois Department of Transportation) record speed, occupancy and flow rate approximately every 5 minutes. Likewise, travel times on toll roads between two I-PASS toll booths (operated by Illinois Toll Authority) are obtained from in-vehicle transponders, and subsequently aggregated and written into database every 5 minutes. About 825 loop detectors and 174 I-PASS detectors (each I-PASS detector corresponds to an origin-destination pair of toll booths) from GCM database are used in this study (see Figure 5.1). GCM contains a larger number I-PASS detectors, many of which have a bound outside of Illinois and therefore were excluded. Malfunctioned detectors and those with duplicated identifications (in the original GCM database, most detectors have two different IDs) were also excluded. In this research, the loop detector data collected from 10/10/2004 to 10/11/2008, and the I-PASS detector data from 10/9/2004 to 7/3/2008, are employed.

5.1.1 Raw data

Figure 5.3 shows the raw data reported at loop detector IL-TSC-EISENHOWER-E-3029 on July 2$^{\text{nd}}$, 2007 (Monday), whose location in the CMAP network can be founded in Figure 5.2. The data suggest that the freeway section is subject to heavy congestion throughout the day. The time-of-day profiles of speed and occupancy well match each other, with

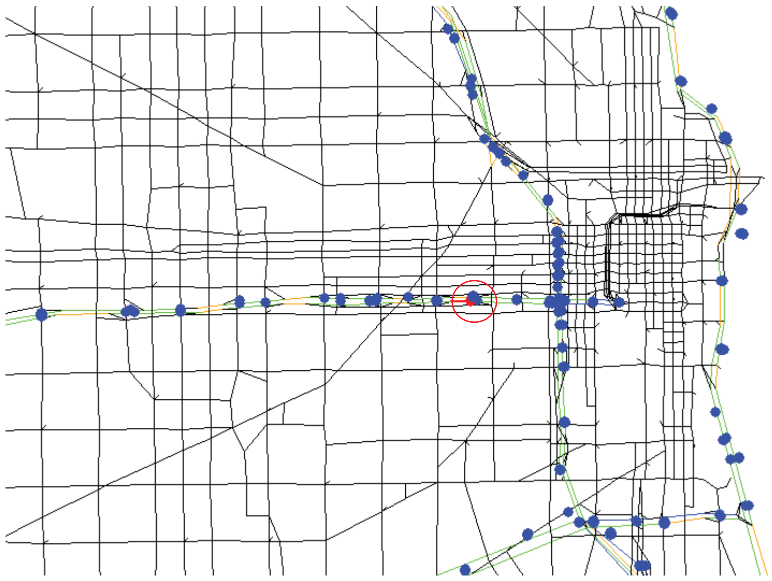


Figure 5.2: The location of loop detector IL-TSC-EISENHOWER-E-3029

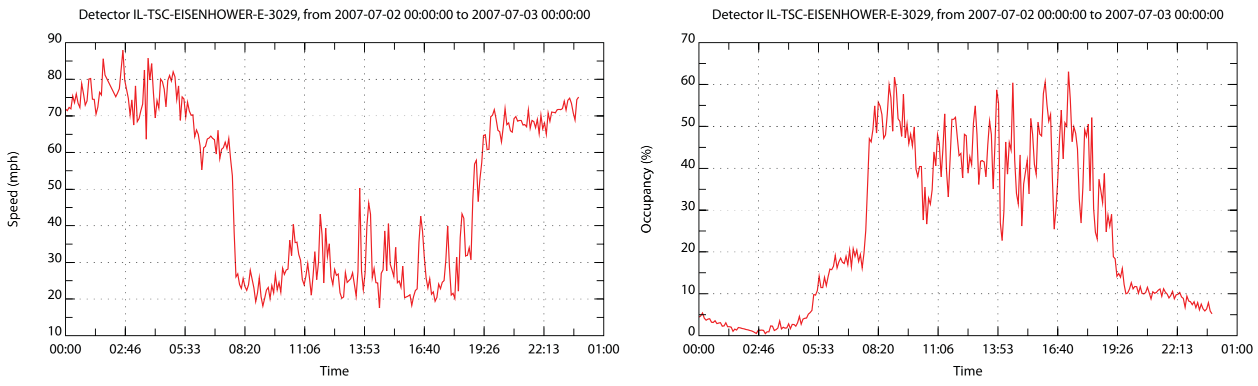


Figure 5.3: Speed and occupancy reported at loop detector IL-TSC-EISENHOWER-E-3029 on 2007/07/02

lower speeds corresponding to higher occupancy, as one would expect. For further verification, Figure 5.4 reports the relationships between speed and occupancy and volume and occupancy at detector IL-TSC-EISENHOWER-E-3029, only using morning peak period data collected on weekdays in Spring (in total there are 11,460 data points). These relationship are perfectly aligned with basic traffic flow theory, which predicts that 1) speeds decrease rapidly as occupancy (density) grows beyond certain threshold (known as critical density); 2) volume grows linearly with speed until it reaches the capacity near the critical occupancy; and 3) the volume-occupancy relationship can be approximately depicted by a triangle. Figure 5.5 illustrates the distribution of travel speeds at the same

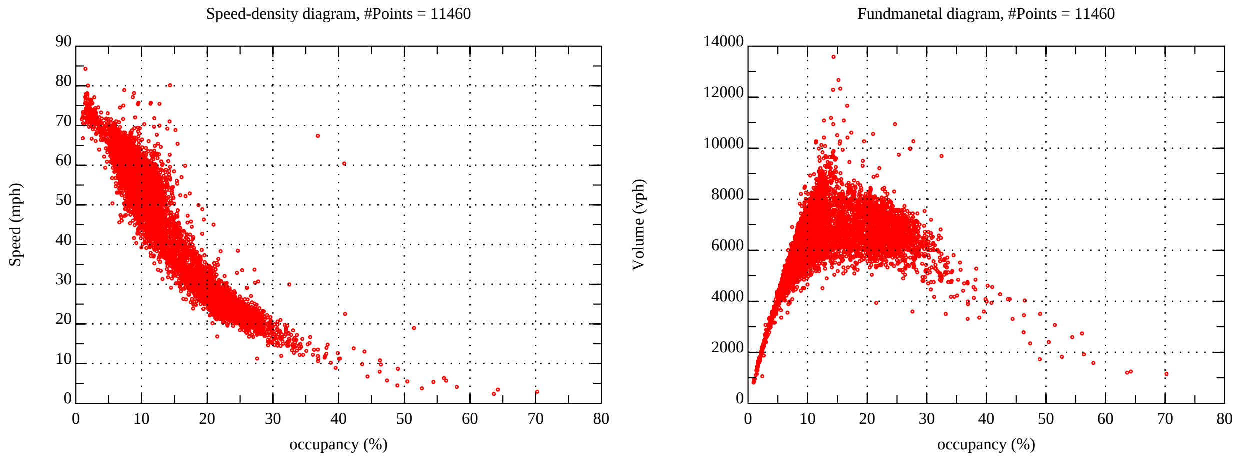


Figure 5.4: Relationships between traffic quantities reported at loop detector IL-TSC-EISENHOWER-E-3029. Morning period (6-10 am), Weekday, Spring (months of March, April and May)

loop detector. As shown, the speed has a strong peak around the free flow value (about 60 mph) but has a long and distinguished tail associated with rush hour congestion. The average speed for that period is about 52 mph and the standard deviation is about 13 mph.

Figure 5.6 reports a sample of raw travel time data for an I-PASS detector (IL-TSC-184), which covers a stretch of Eisenhower highway (from Halsted to East-West streets, see Figure 5.7). The figure reveals two distinct rush hour periods, with a stronger evening peak (with travel times almost tripped comparing with off-peak numbers). The empirical distribution of travel time for the above I-PASS detector is plotted in Figure 5.8. In total, these distributions are produced based on 8210 valid observations that were made during morning periods on weekdays in Spring. Note that the shape of the probability density function is similar to that of a Gamma distribution. The average travel time during the morning period for that stretch of freeway is about 28 minutes, with a standard deviation of 10 minutes.

5.1.2 Obtain link travel times

Ultimately, the underlying inputs to reliable routing are travel time distributions at link level. To construct these distributions, travel times on a link for any of the 288 intervals

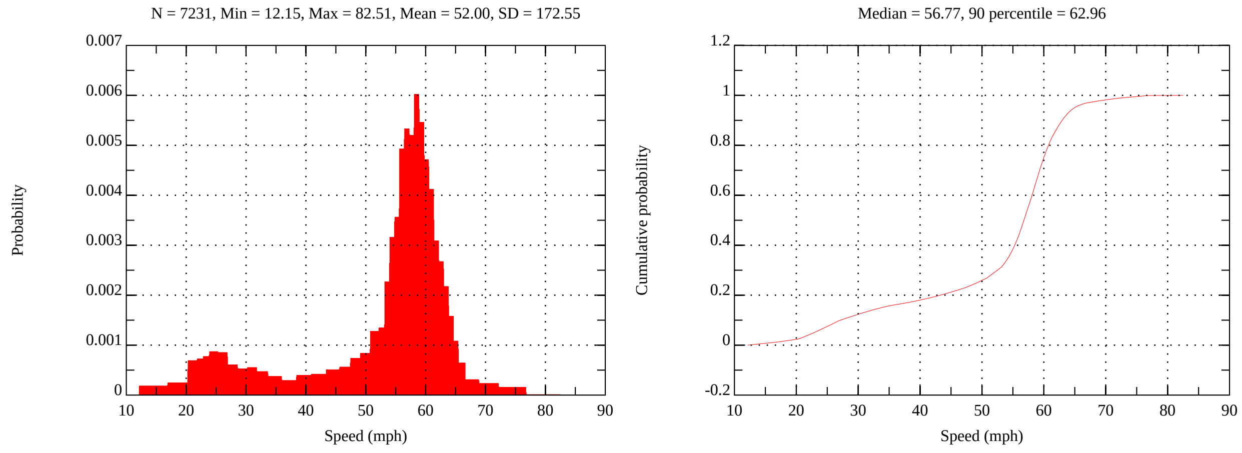


Figure 5.5: Speed distribution at loop detector IL-TSC-EISENHOWER-E-3029. Morning period (6-10 am), Weekday, Spring (months of March, April and May)

$(12/hr \times 24hr/day)$ in any day covered by the data collection period must be derived from loop detector or I-PASS data.

We first need to establish a set of links that are "covered" by either I-PASS detector or loop detector. To determine which link in the CMAP network is associated with a loop detector, the coordinates (longitude and latitude) of the detector (available in the GCM database) are used to find the closest freeway link. A computer program was developed to automate the process, but manual corrections were often found to be necessary. Finding the I-PASS covered links requires more work, which consists of two major steps. In the first, the starting and ending points of the I-PASS detector are located in the CMAP network. Similarly, this work requires some manual maneuvers after the computer did the first cut. The second step identifies and marks all links used by the fastest (not shortest) path connecting the two points. Thus, vehicles that pass two I-PASS toll booths in sequence are assumed to always stay on the toll roads, which in most cases constitute the fastest alternative. Note that a link in the CMAP network may be covered by more than one loop detectors. In total, 765 out of 44331 links are covered in one way or another, as shown in Figure 5.9.

For links covered by loop detector(s), the recorded speed in a 5-minute interval can be used to estimate link travel time for the corresponding interval, i.e.,

$$\tau_a^d(t) = l_a / v_a^d(t) \tag{5.1}$$

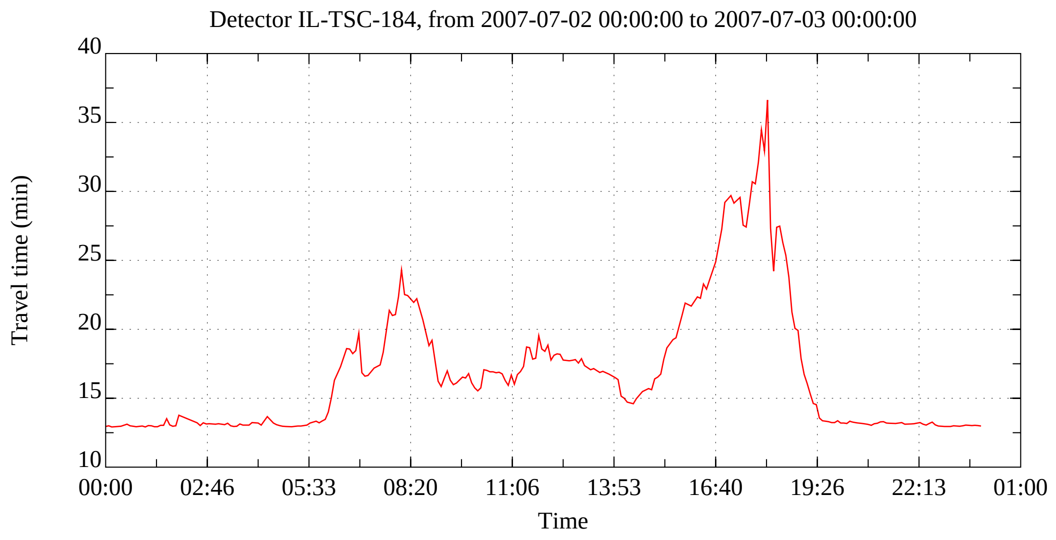


Figure 5.6: Travel time reported for I-PASS detector IL-TSC-184 on 2007/07/02

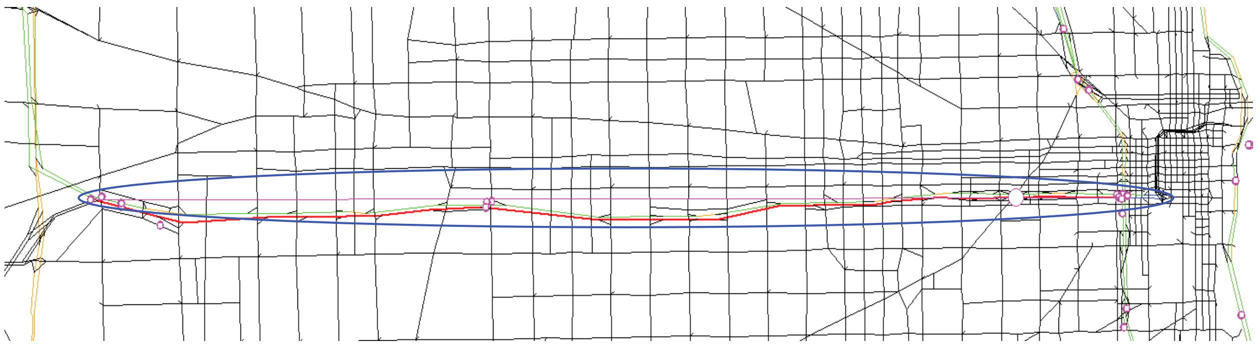


Figure 5.7: The location of I-PASS detector IL-TSC-184

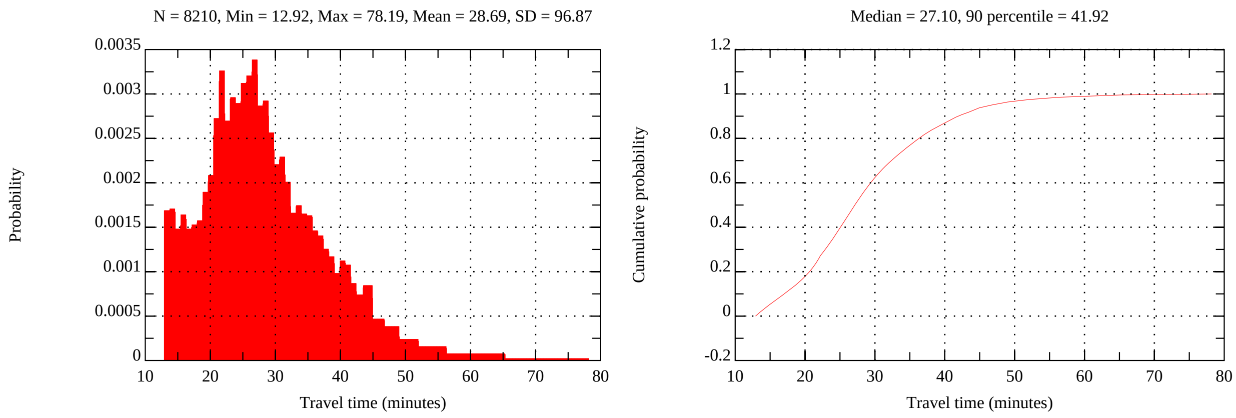


Figure 5.8: Travel time distribution for I-PASS detector IL-TSC-184. Morning period (6-10 am), Weekday, Spring (months of March, April and May)

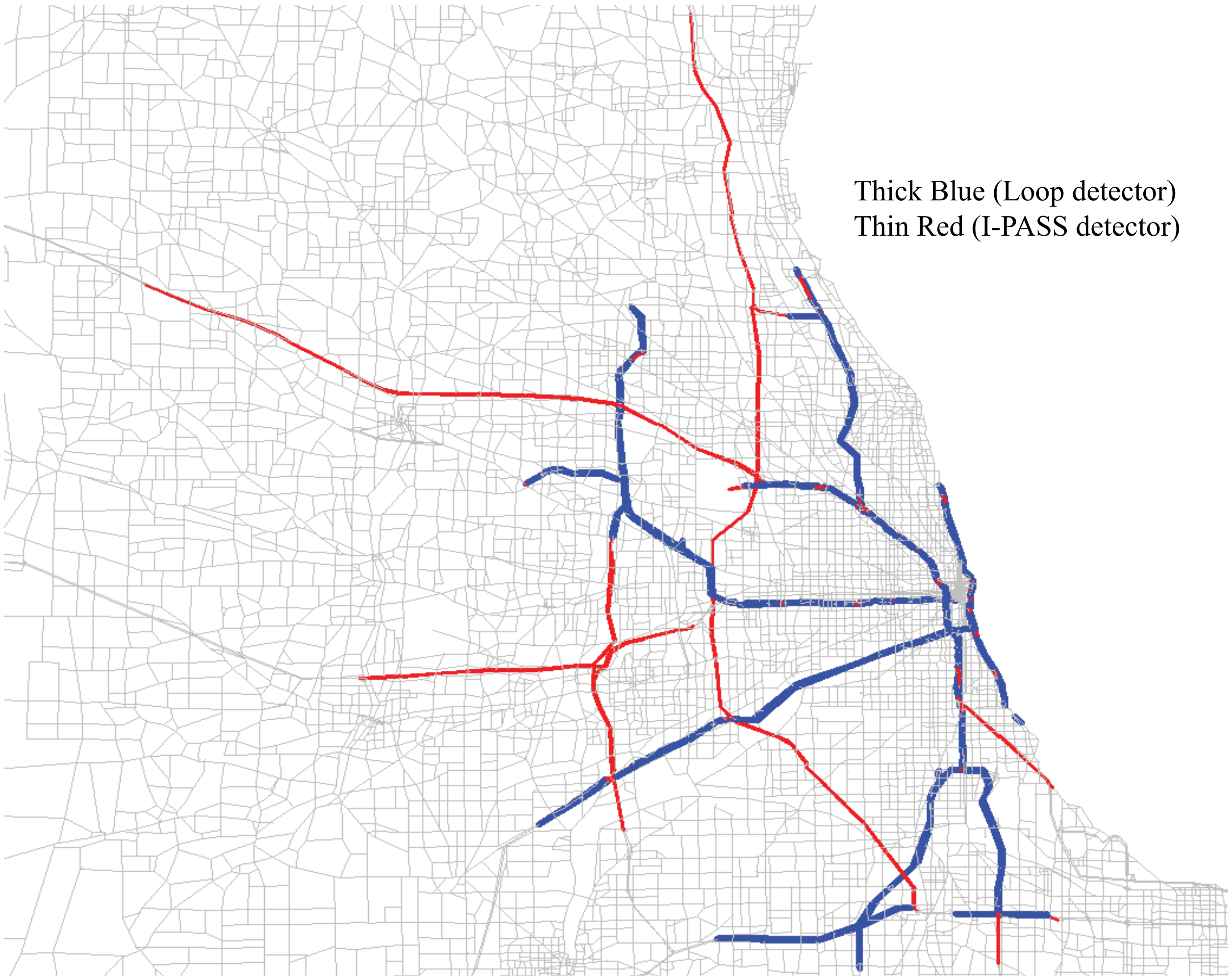


Figure 5.9: Covered links in the CMAP network (total covered links = 765)

where $\tau_a^d(t)$ and $v_a^d(t)$ be travel time and speed on link a recorded by detector d for time interval t, and l_a is the link length. If for an interval, a link contains more than one recorded travel time, the arithmetic average of calculated travel time values is taken as the nominal link travel time, that is,

$$\tau_a^d(t) = \frac{\sum_{d \in D(a)} l_a / v_a^d(t)}{|D(a)|} \tag{5.2}$$

where $D(a)$ is the set of loop detectors associated with link a at a given time.

As for I-PASS detectors, we need to estimate link travel times on covered link based on the recorded path travel times for a given time period. This is a difficult exercise for two reasons. First, how the travel delays (if there is any) experienced on a path may be *spatially* distributed is unknown. Second, an I-PASS record tagged by one time interval might contribute link travel times at other time intervals [1]. It is hard to solve either problem unless further information is available, such as supplementary loop detector data. For simplification, we assume that the path travel times are distributed to links proportional to their lengths, that is

$$\tau_a^i(t) = \frac{l_a}{\sum_{a \in k^{rs}} l_a} c_k^{rs}(t) \tag{5.3}$$

where k^{rs} denotes the shortest path connecting nodes r (the starting node of the link associated with the origin toll booth) and s (the ending node of the link associated with the destination toll booth), and $c_k^{rs}(t)$ is the recorded travel time on the path for time interval t. While this simple treatment would certainly introduce errors, we note that the magnitude of errors may be alleviated when multiple I-PASS records are available for the same stretch of toll roads. Equation 5.3 also implies that the travel time on a path at one time interval contributes to its covered links for the same interval. This shortcoming is not as serious as it sounds, since eventually the travel time data will be aggregated on a period that last several hours. That is to say, as long as the misplaced link travel times do not go into a wrong period (which is certainly possible but is much less severe), they will not seriously distort the aggregated distributions. To summarize, the travel time on

[1] Note that the time interval that identifers an I-PASS record must be tied to either the entry (origin) or exit (designation), since the travel times between most I-PASS toll booths are longer than 5 minutes.

link a at time t is given by

$$\tau_a(t) = \begin{cases} \tau_a^d(t) & \text{if loop data are available} \\ \tau_a^i(t) & \text{if I-PASS data are available} \end{cases} \tag{5.4}$$

5.1.3 Construct distributions

Once link travel times are obtained, the empirical distributions can be constructed using the following procedure.

Step 1 Find $L_a = \min\{\tau_a(t), \forall t \in \Lambda\}, U_a = \min\{10 l_a / v_a^0, \max\{\tau_a(t), \forall t\}\}$, where Λ is a set of valid time intervals, and v_a^0 is free flow speed (or speed limit) on link a.

Step 2 Divide $[L_a, U_a]$ into M intervals, and let $\delta_a = (U_a - L_a)/M$. Find the set $S_m = \{\tau_a(t) | \forall t \in \Lambda, (m-1)\delta a \leq \tau_a(t) < m\delta\}, \forall m = 1, ..., M$

Step 3 Obtain the probability mass for each interval m using

$$P_m = \frac{|S_m|}{|\Lambda|}$$

It is noted that link travel time distribution may be affected by various factors, such as time-of-day and seasonal effects. Consequently, one should consider a different reliable routing decision for rush hour and off-peak period. To address this issue, the travel time data are disaggregated according to three key factors: time-of-day, day-of-week and season. Specifically, each day is divided into four periods, namely, morning peak period (6 am - 10 am), mid-of-day period (10 am - 15 pm), evening peak period (15 pm - 20 pm) and off-peak period (20 pm - 6 am). Days in a week are first grouped into weekends and weekdays. In addition, Friday, Saturday and Sunday are separated to form individual groups because the travel patterns on these days are subject to large variances. Finally, a year is grouped into Spring (months of 3, 4 and 5), Summer (months of 6, 7 and 8), Fall (months of 9, 10, 11) and Winter (months of 12, 1, 2). For each of the three factor, an additional group is added to address the case of no-segmentation. For instance, the segmentation for time-of-day contains 5 instead of 4 groups: morning peak, mid-of-day, evening peak, off-peak and whole-day (no segmentation for time-of-day). Therefore, in total, there are $5 \times 6 \times 5 = 150$ possible combinations. Accordingly, we generate 150 different distributions for all the 765 covered links and store them in a MYSQL database.

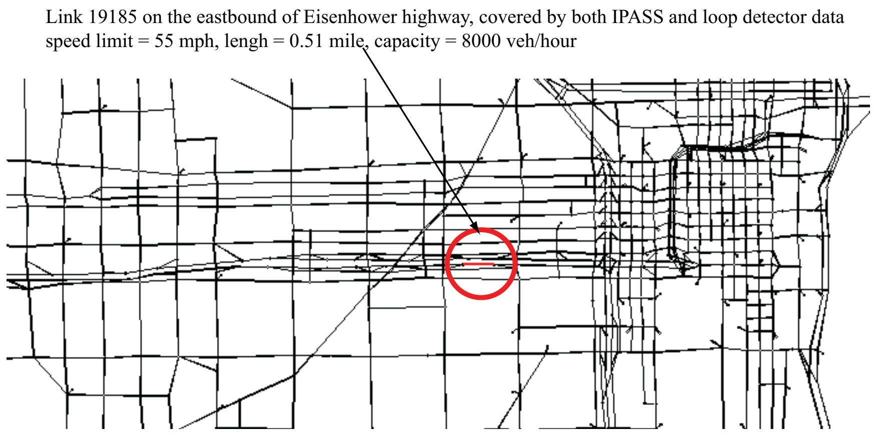

Link 19185 on the eastbound of Eisenhower highway, covered by both IPASS and loop detector data
speed limit = 55 mph, lengh = 0.51 mile, capacity = 8000 veh/hour

Figure 5.10: Location of the sample link (ID = 19815)

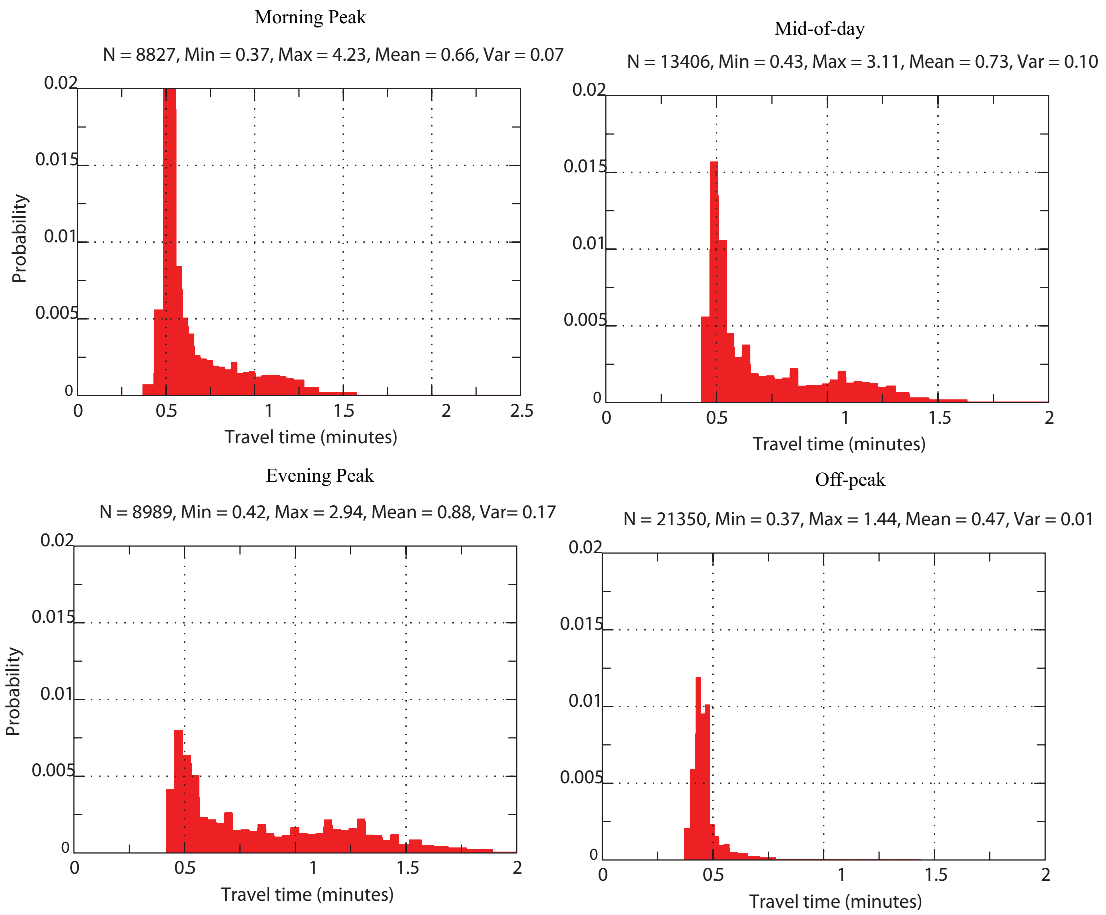


Figure 5.11: Comparison of link travel time distribution at different time-of-day periods (Link 19815, Spring, Weekday)

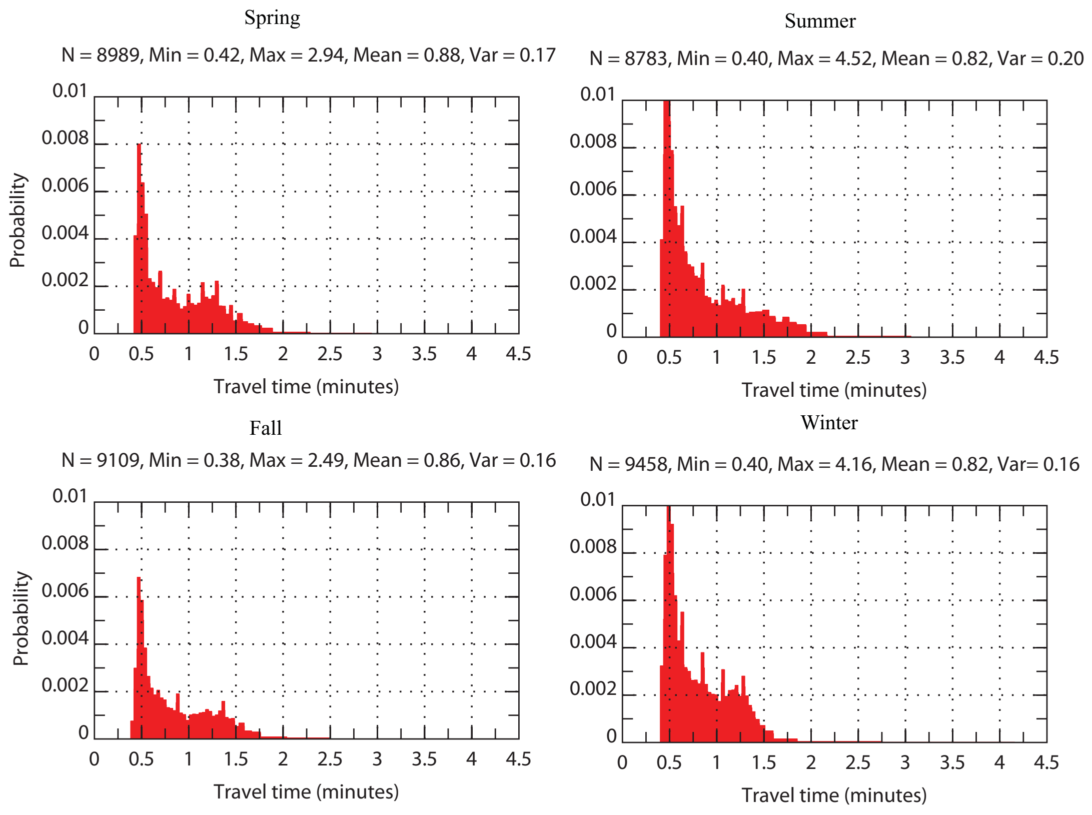


Figure 5.12: Comparison of link travel time distribution in different seasons (Link 19815, Weekday, Evening peak)

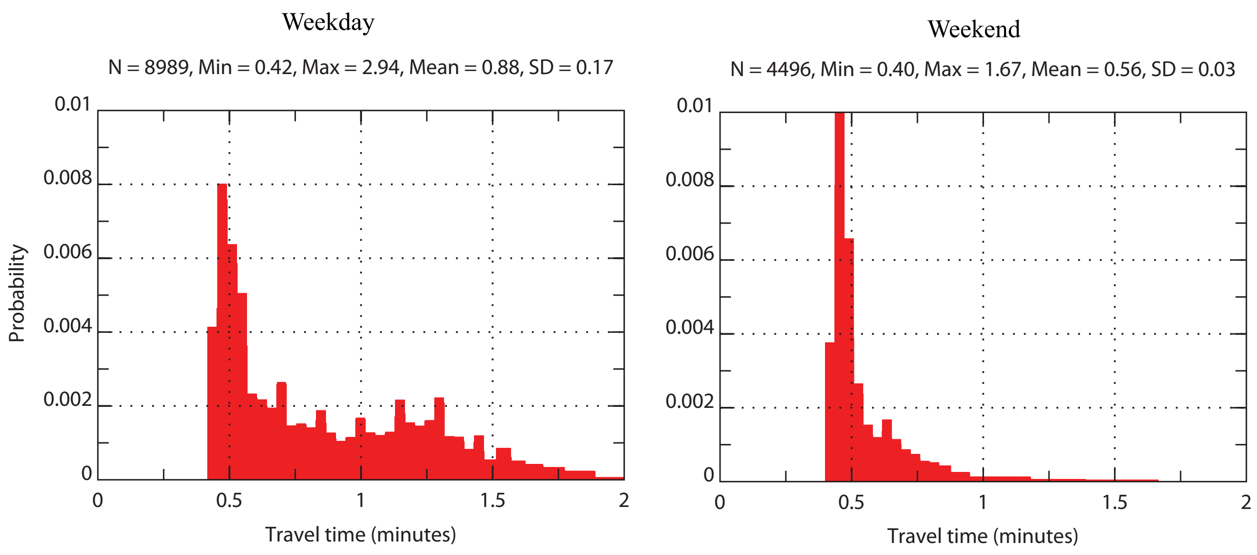


Figure 5.13: Comparison of link travel time distribution on weekdays and weekends (Link 19815, Spring, Evening peak)

An example of generated link travel time distributions is given below. Figure 5.10 identifies the link in the CMAP network. A comparison of link travel time distributions for the four periods is given in Figure 5.11. The results suggest that the most congested period is the evening peak, followed by mid-of-day, morning peak and off-peak. Also, the variance of the distributions increases as the road becomes more congested. For example, the standard deviation for off-peak and evening peak is 0.1 and 0.4 minutes, respectively.

Figure 5.12 compares the travel time distributions for the same link in different seasons on weekday and during evening peak period. As shown, although the seasonal effect does not affect distribution as strongly as time-of-day, it is not negligible. Interestingly, the data suggest that the travel time variance in the summer is about 25% higher than other season, which may be linked to construction work.

Finally, a comparison between weekdays and weekends for the same link is reported in Figure 5.13. As expected, the results suggest that the road is subject to much higher mean travel time and larger variances on weekdays.

5.2 Data for arterial and local streets

No observations are available for arterial and local streets in the CMAP network. Consequently, the travel time distributions on these links have to be estimated indirectly. The estimation process involves two main steps: select an appropriate functional form, and estimate mean and variance.

Travel time on freeways and arterial streets is known to closely follow a Gamma distribution (e.g. Polus 1979). Figure 5.11 provides further confirmation. Gamma distributions have also been adopted in various studies of stochastic routing problems (e.g. Fan et al. 2005*a*, Nie & Wu 2009*b*). Therefore, this study adopts Gamma distribution to describe the travel time distribution on arterial and local streets. The probability density function of a Gamma distribution is

$$f(x) = \frac{1}{\theta^{\kappa}\Gamma(\kappa)}(x-\mu)^{\kappa-1}e^{-(x-\mu)/\theta}; x \geq \mu, \theta, \kappa \geq 0 \tag{5.5}$$

where θ is the *scale parameter*; κ is the *shape parameter*; μ is the *location parameter*; and $\Gamma(\cdot)$ is the Gamma function which takes the following form

$$\Gamma(z) = \int_0^{\infty} t^{z-1}e^{-t}dt$$

Having selected the functional form, we proceed to show how to estimate the three parameters required in a Gamma distribution. We first note that the mean and variance of a Gamma distribution are $\kappa\theta$ and $\kappa\theta^2$, respectively. Thus, if we know mean (denoted as u), variance (denoted as σ^2) and μ, then κ and θ can be obtained by

$$\theta = \frac{\sigma^2}{u-\mu}, \kappa = (\frac{u-\mu}{\sigma})^2 \tag{5.6}$$

The CMAP travel demand model (CMAP 2006) is used to estimate congested travel times on arterial streets. Note that the planning model is designed to capture average traffic conditions in the network for the designated time period on a typical weekday. The CMAP travel demand model represents a classical four-step process of *trip generation*, *trip distribution*, *mode choice*, and *traffic assignment*, with considerable modifications used to enhance the distribution and mode choice procedures. The original CMAP model divides a day into eight periods: off-peak (8 PM - 6 AM), pre-morning-peak (6-7 AM), morning-peak (7 - 9 AM), post-morning-peak (9-10 AM), mid-of-day (10 AM - 2 PM), pre-evening-peak (2 - 4 PM), evening-peak (4 - 6 PM), and post-evening-peak (6 - 8 PM). Note that in GCM data peak periods combine the pre and post periods defined in the CMAP model. For simplification, the assignment results for the peak periods (morning and evening) in the CMAP model are used to represent those from pre to post peak periods. Specifically, each link obtains from the CMAP model a mean travel time for each of the four predetermined GCM periods: morning peak, mid-of-day, evening peak and off-peak.

To estimate the mean (u), variance (σ^2) and the location parameter (μ) is less straightforward, since they are not readily available from the travel demand model. We postulate that the mean and variance of travel times on a link are related to its free flow travel time τ^0 and the level of congestion $\rho = \tau - \tau^0$, where τ is travel time from traffic assignment (note that the subscript a is suppressed for simplicity). This relationship may be estimated from freeway data using statistical models. The simplest linear regression model reads

$$u = a_1\tau^0 + b_1\rho + c_1 \tag{5.7}$$

$$\sigma = a_2\tau^0 + b_2\zeta\rho + c_2 \tag{5.8}$$

where a_1, b_1, c_1, a_2, b_2 and c_2 are coefficients to be estimated from linear regression. ζ is a predetermined parameter to account for the fact that the existence of signal control may increase variances. $\zeta = 1$ if no signal exists on the link; otherwise ζ is taken from a uniform distribution between $[1.1, 1.3]$. For all 765 links covered by GCM data, u and σ can be obtained from the empirical distribution and ρ is known from the CMAP travel demand model. Thus, a linear regression can be performed to determine the coefficients, which in turn are employed to estimate u and σ for arterial streets. We note that a linear model is needed for each of the four time-of-day periods.

A similar linear model can be constructed to estimate the location parameter μ, which delineates the smallest possible travel time on a link. We note that μ is likely to be smaller than τ_0 because motorists may drive well beyond the speed limit or the nominal "free-flow travel speed". Moreover, it seems reasonable to assume that the level of congestion

Table 5.1: Results of the linear regression for the mean-variance model (Equations 5.7-5.8) and the location model (Equation 5.9)

time-of-day	Variance Model				Mean Model				Location Model		
periods	a_1	b_1	c_1	R^2	a_2	b_2	c_2	R^2	a	b	R^2
AM PEAK	0.309	0.870	0.580	0.444	1.127	0.546	-2.056	0.910	0.843	-4.106	0.958
PM PEAK	0.368	0.685	2.967	0.400	1.143	0.563	0.336	0.872	0.860	-3.533	0.964
MIDDAY	0.283	1.076	2.040	0.346	1.100	0.630	-1.145	0.889	0.857	-3.608	0.956
OFF PEAK	0.178	0	-1.031	0.516	1.043	0	-5.854	0.907	0.831	-5.257	0.937

does not affect μ. Thus, the linear model used to estimate μ is given by

$$\mu = a\tau^0 + b \tag{5.9}$$

The linear regression results for each of the four time-of-day periods are given in Table 5.1. As shown, the mean model and the location model (Equation 5.9) fit the data rather well (high R^2). However, the fitness of the variance model is not impressive. We tried, without much success, to introduce various forms of non-linearity into the model, such as using the variance (σ^2) instead of the standard deviation on the left hand side of Equation (5.8), or consider $(\tau - \tau_0)^2$ on the right hand side. Apparently, the travel time variances are affected by many other factors not included in the simple linear model. We leave a more in-depth investigation of the variance model to the furture research. Finally, we note that the coefficient of the congestion index is near zero in both mean and variance models for off-peak period because congestion is negligible in that period. Figure 5.14 reports distributions on a sample link (ID=18548, located on the northbound Columbus Dr. in the Chicago downtown). The results show that the street is slightly more congested during the mid-of-day and evening peak periods.

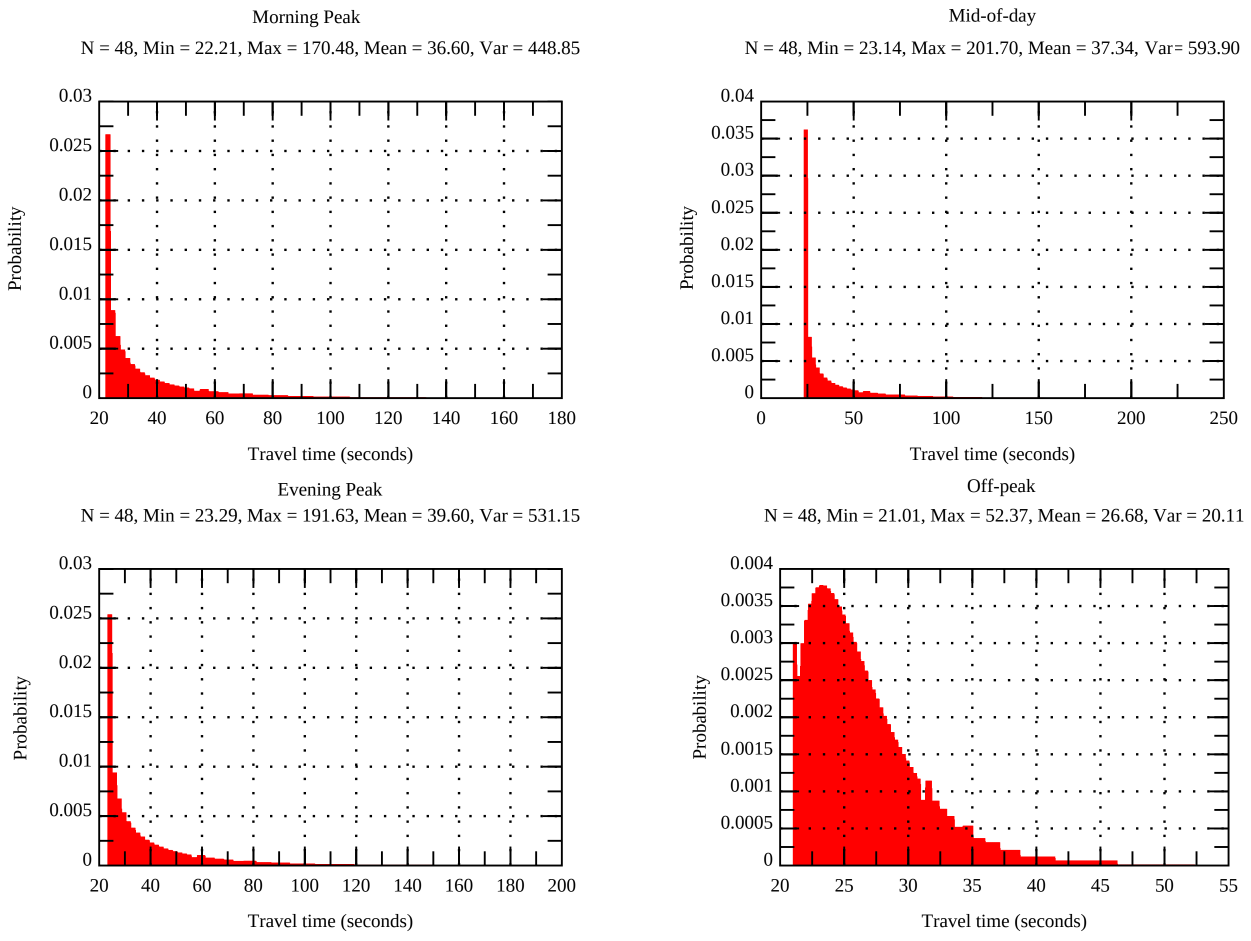


Figure 5.14: Comparison of distributions of link travel time at different time-of-day periods (Link 18548)

Chapter 6

Numerical Experiments

Numerical experiments are presented in this section to show the usefulness of reliable route guidance and the feasibility of the existing algorithm in solving the real-size problems. For the first, we consider several real-world routing examples. The algorithm FSD-LC was coded using MS-C++ and tested on a Windows XP x64 Workstation with two Xenon 3.0GHz CPUs and 8GB RAM.

6.1 From Chicago downtown to O'hare international airport (ORD)

In this experiment, the intersection of Wabash St. and Washington St. is selected to represent the Chicago downtown, and the airport is represented by the end of I-190, the highway that serves the terminals. The most obvious choice for this routing problem, which is also suggested by both Google Map and Yahoo Maps, is to use freeway I-90/94 and I-90. Our results agree with this popular routing policy in general but have interesting discrepancies. For the mid-of-day period, most FSD-admissible paths always heavily use I-90 and I-90/94, and the differences between these paths are trivial, see Figure 6.1.

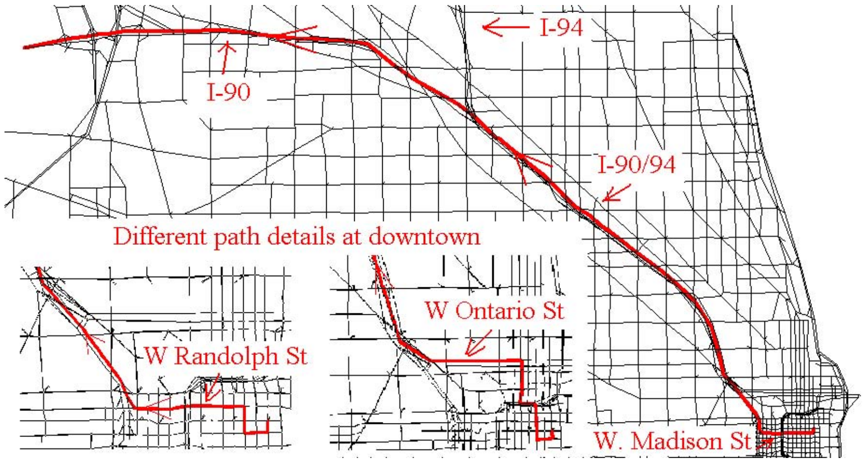
For the morning peak period, however, the reliable route guidance suggests that motorists should avoid I-90/94 and use an arterial street (N. Milwaukee Ave.) instead, if they wish to have an on-time arrival probability higher than 44% (to airport) or 62% (from airport) on-time arrival probability. To arrive at the airport with 95% probability, for example, the path in Figure 6.2(a) requires a time budget of 33 minuets 57 seconds while the path mostly using I-90 and I-90/94 needs 37 minutes and 18 seconds. The reliable route guidance thus leads to a 10% saving in travel budget. In the evening peak of weekdays, motorists who drive from the airport to the city are recommended to avoid I-90 until they pass the diverge of I-90 and I-94 (see Figure 6.3(a)). For 95% on-time arrival probability, the path in Figure 6.3(a) requires a time budget of 34 minutes and 30 seconds, while the path in Figure 6.3(b) needs 39 minutes and 38 seconds. In both periods, our results suggest that avoiding the entire or part of the popular freeways will help motorists budget less time for better reliability.

We note that the path given in Figure 6.3(b) is actually the best for 50% on-time arrival probability (i.e. the average performance). At this probability, a motorisit using the path only need to budget 34 minutes 17 seconds for travel. As a comparison, the more reliable path in Figure 6.3(a) needs a slightly higher budget (34 minutes 34 seconds) for 50% probability.

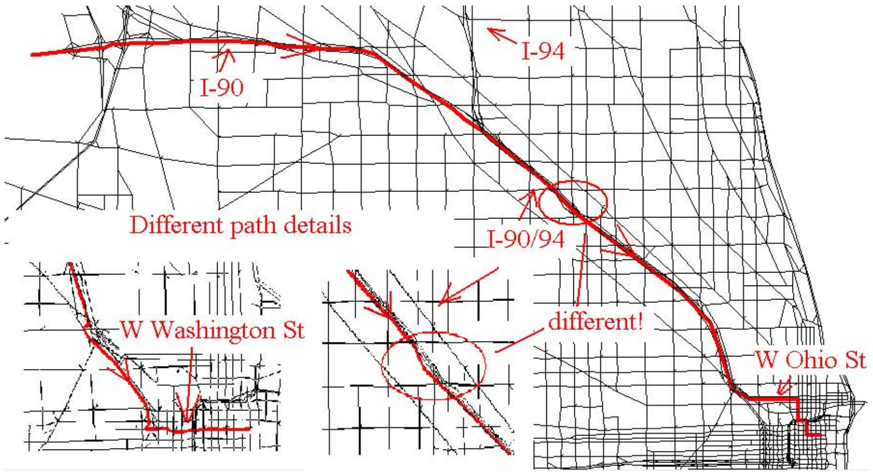
Figure 6.4 reports the CDFs of travel times on the FSD-admissible paths. Note that the CDFs during the mid-of-day period are very close to each other. A close look reveals that all these admissible paths heavily use I-90/94 and I-90, with negligible topological differences. On the other hands, the CDFs for the morning and evening peak periods are quite different, which are related to the substantial topological difference of the admissible paths.

6.1.1 From the west suburbs to Chicago downtown

This experiment considers the routing problems from the west suburbs to Chicago downtown as well as the reverse direction. The purpose of the experiment is to test the reliability of I-290, a popular route from the west suburb to Chicago downtown. The intersection of Michigan Ave. and Randolph Dr. is selected to represent the downtown, and the intersection of W. 31st St. and N. Brainard Ave. is selected to represent the west suburbs. A popular choice for this routing problem is to go north along N. La Grange Rd (the nearest entrance to I-290) and then to drive 13 miles along I-290 East to the downtown, as shown in Figure 6.5(a). This path is also suggested by both Google and Yahoo Maps. However, our experiments show that this path turns out to be too risky. For example, it is shortest only for on-time arrival probability no higher than 7% in the mid-of-day

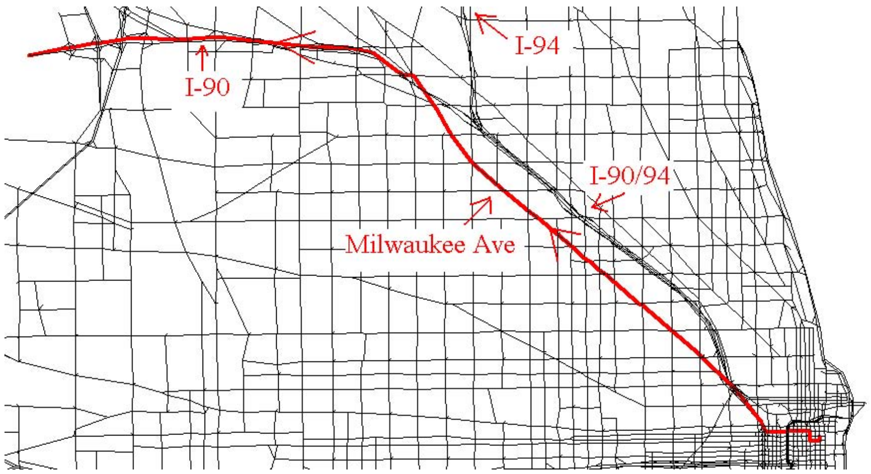


(a) Typical path from downtown to ORD

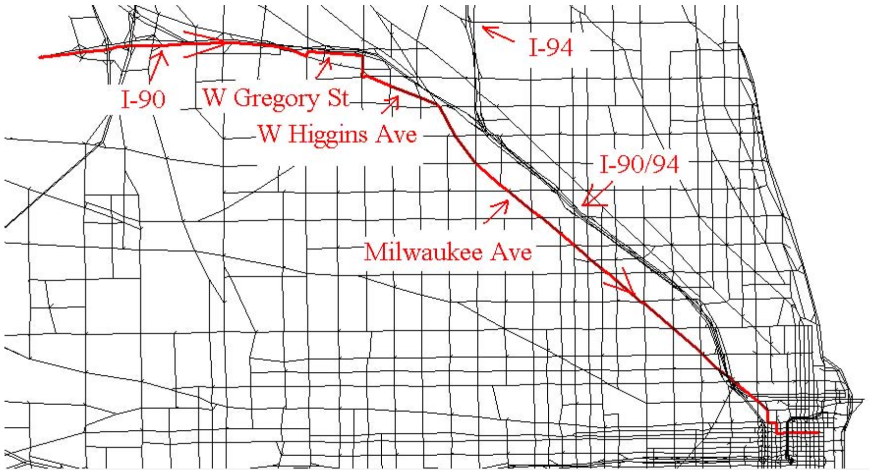


(b) Typical path from ORD to downtown

Figure 6.1: FSD-admissible paths between Chicago downtown and ORD during the mid-of-day period of weekdays

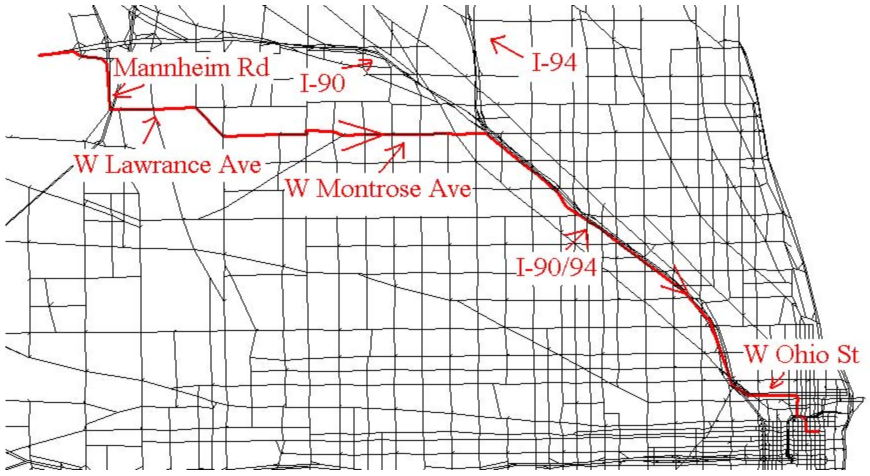


(a) from downtown to ORD

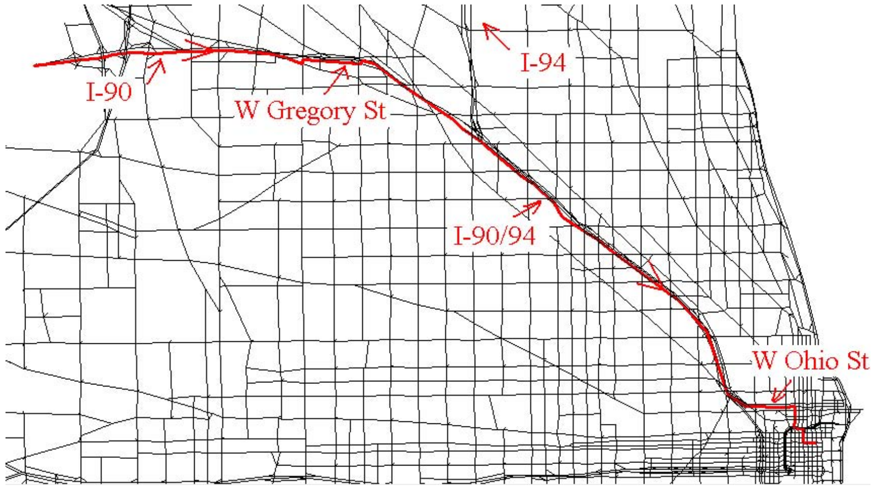


(b) from ORD to downtown

Figure 6.2: Shortest paths between Chicago downtown and ORD) during the morning peak of weekdays (desired on-time arrival probability = 95%)

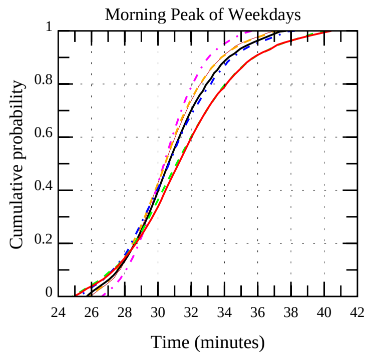


(a) 95% on-time arrival probability

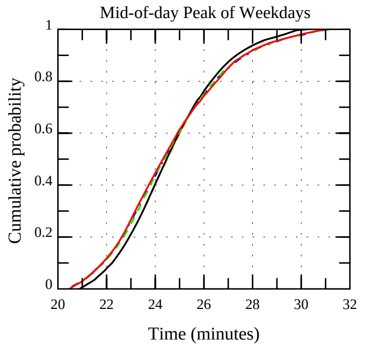


(b) 50% on-time arrival probability

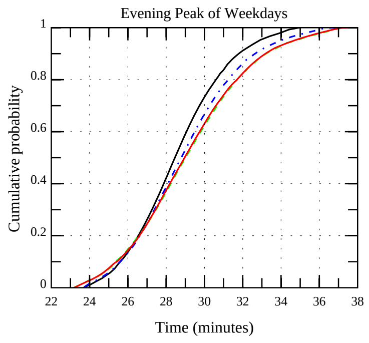
Figure 6.3: Shortest paths from ORD to Chicago downtown during the evening peak of weekdays



(a)



(b)



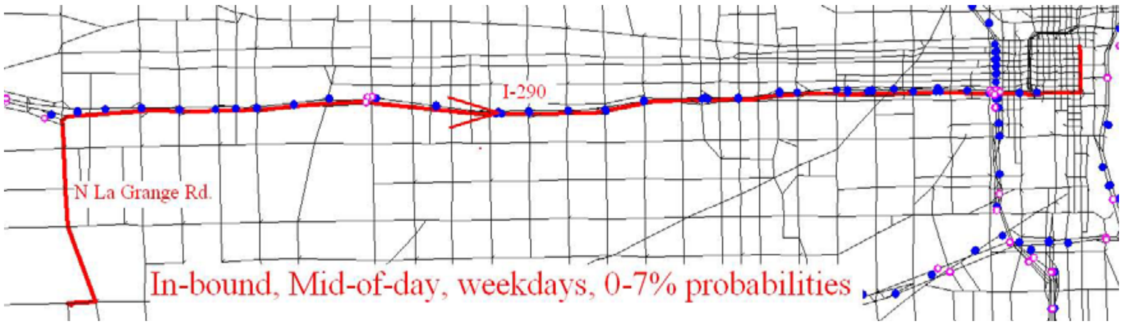
(c)

Figure 6.4: Cumulative density functions (CDFs) of travel times on the FSD-admissible paths for trips from Chicago downtown to ORD during different time periods

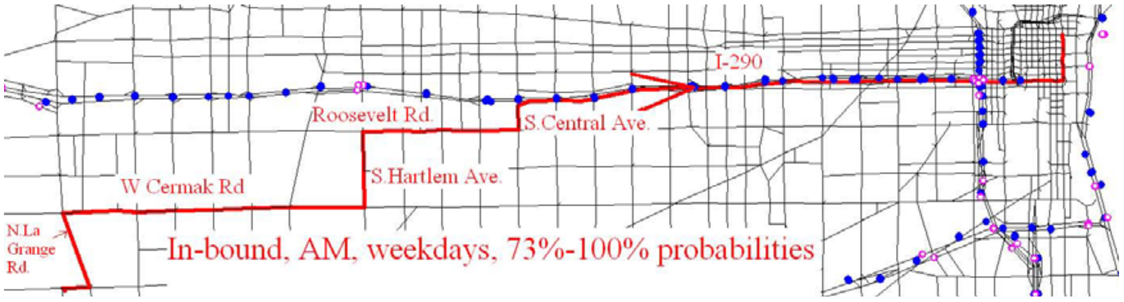
period, and it is even not FSD-admissible in the morning and evening peak period. For the morning peak period, motorists are suggested to use arterial streets (N. La Grange Rd., W. Cermak Rd, S. Hartlem Ave., Roosevelt Rd and S. Central Ave.) and then to enter I-290 East through S. Central Ave., as shown in Figure 6.5(b), if they needs an on-time arrival probability higher than 73%. The driving distance on I-290 East is only around 7 miles: almost half of this path is made up of arterial and local streets. Moreover, Figures 6.5(b)-6.5(d) show that the motorists should enter I-290 East later if they prefer a higher on-time arrival probability in weekdays. For example, if following the path in Figure 6.5(c), motorists enter I-290 East 4 miles earlier than following the path in Figure 6.5(b), but it causes 5% more in travel time if they want to arrive at the destination with 95% on-time arrival probability, for the morning peak period.

Now consider the reverse direction: from Chicago downtown to the west suburbs. Both Google and Yahoo maps suggest using I-290 West until reaching Exit 17B (N. La Grange Rd.), as shown in Figure 6.6(a). This path, however, is also risky in weekdays, especially for the morning and evening peak periods. It is shortest only when the on-time arrival probability is no higher than 38% (morning peak) or 34% (evening peak). The reliable route guidance suggests motorists avoiding I-290 largely if they prefer a high on-time arrival probability. For example, as shown in Figure 6.6(b), motorists are told to leave I-200 West at Exit 27A (S. Homan Ave.) instead of Exit 17B (N. La Grange Rd.) for the morning peak period. It means that the driving distance on I-290 West is only 4 miles rather than 13 miles, but it leads 12% saving in travel time under 95% on-time arrival probability. For the evening peak period, motorists are suggested to leave I-290 West even earlier at Exit 28B, around 2 miles ahead of Exit 27A, so that the travel time can be saved 14% under 95% on-time arrival probability. Figure 6.6(c) shows that a detour on I-55 South can even save time than driving on I-290 West for the morning peak period. For 95% on-time arrival probability, the path in Figure 6.6(c) requires a time budget of 33 minutes and 35 seconds, but the path in Figure 6.6(a) requires 36 minutes 11 seconds. The latter travel time is 8% more than the former. Therefore, the experiments indicate that both directions of I-290 are congested during the peak time periods.

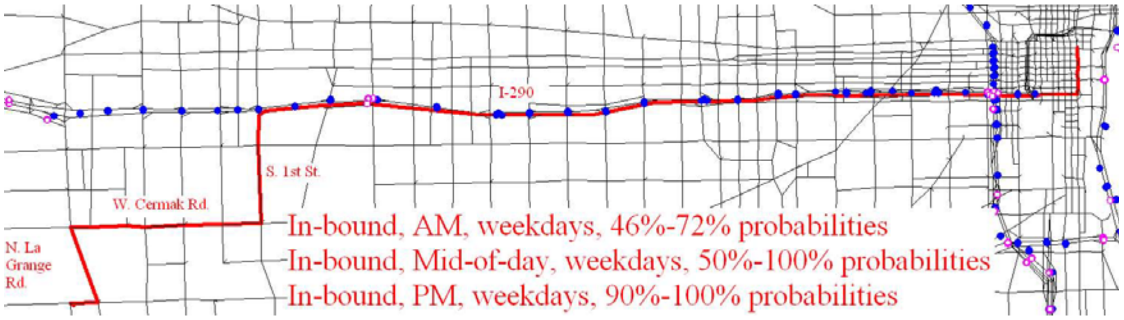
Figure 6.1.1 reports the CDFs of travel time on the FSD-admissible paths for trips



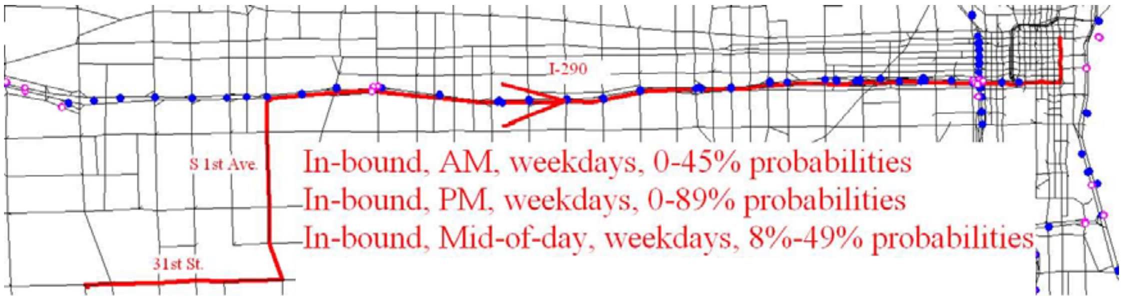
(a)



(b)

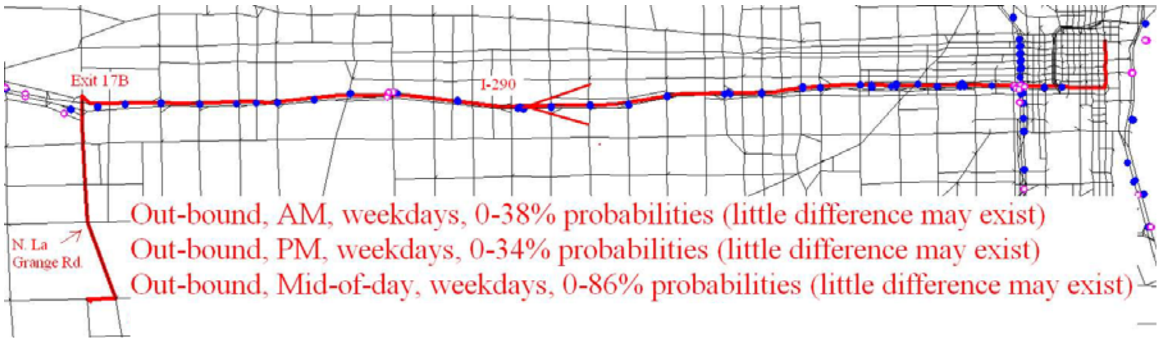

I-290
S. 1st St.
W. Cermak Rd.
N. La Grange Rd.
In-bound, AM, weekdays, 46%-72% probabilities
In-bound, Mid-of-day, weekdays, 50%-100% probabilities
In-bound, PM, weekdays, 90%-100% probabilities

(c)

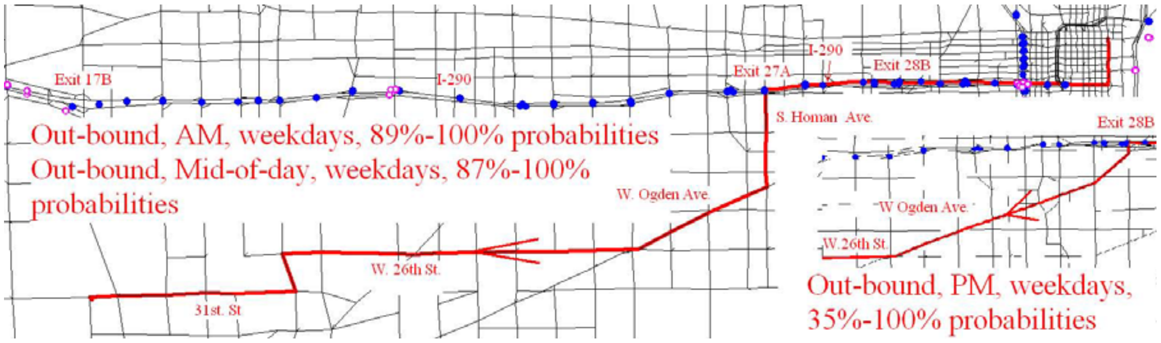

I-290
S 1st Ave.
31st St.
In-bound, AM, weekdays, 0-45% probabilities
In-bound, PM, weekdays, 0-89% probabilities
In-bound, Mid-of-day, weekdays, 8%-49% probabilities

(d)

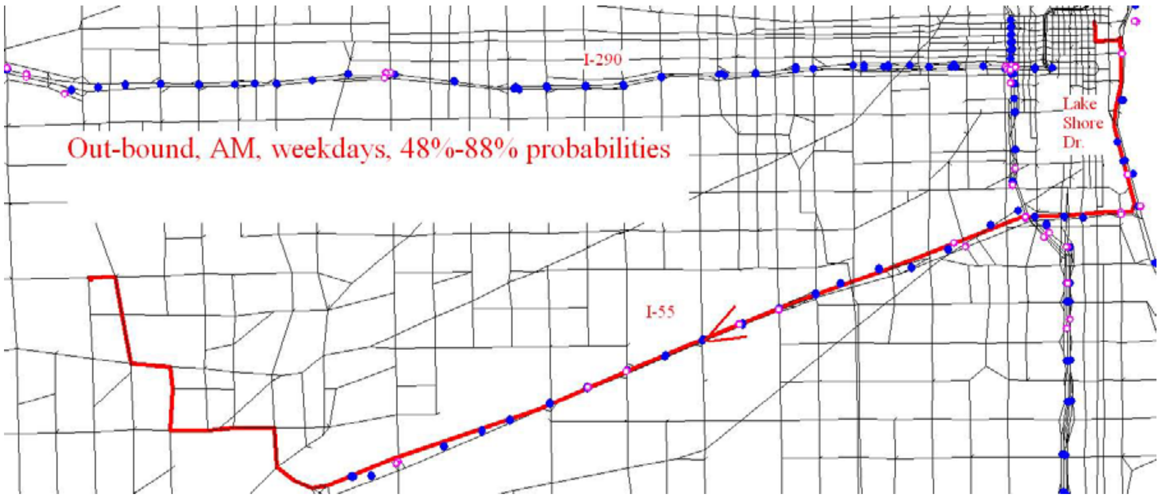
Figure 6.5: Shortest paths from the west suburbs to Chicago downtown


Exit 17B
I-290
N. La Grange Rd.
Out-bound, AM, weekdays, 0-38% probabilities (little difference may exist)
Out-bound, PM, weekdays, 0-34% probabilities (little difference may exist)
Out-bound, Mid-of-day, weekdays, 0-86% probabilities (little difference may exist)

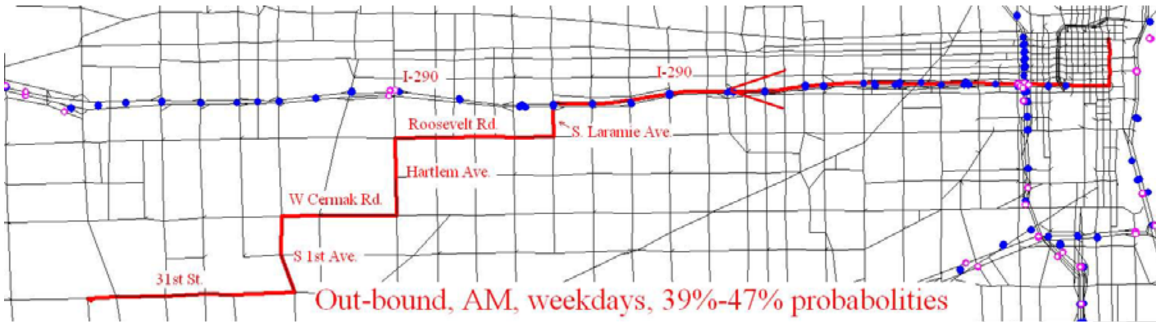
(a)


Exit 17B
I-290
Exit 27A
I-290
Exit 28B
S. Homan Ave.
Exit 28B
Out-bound, AM, weekdays, 89%-100% probabilities
Out-bound, Mid-of-day, weekdays, 87%-100% probabilities
W. Ogden Ave.
W Ogden Ave.
W.26th St.
W. 26th St.
31st. St
Out-bound, PM, weekdays, 35%-100% probabilities

(b)



(c)



(d)

Figure 6.6: Shortest paths from Chicago downtown to the west suburbs

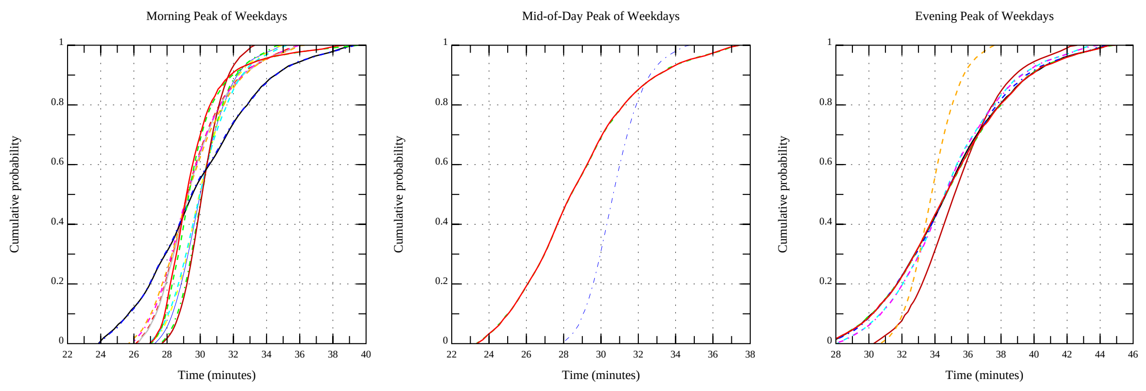


Figure 6.7: Cumulative density functions (CDFs) of travel time on the FSD-admissible paths for trips from Chicago downtown to the west suburbs for the morning, mid-of-day and evening peak periods of weekdays

from Chicago downtown to the west suburbs for the morning, mid-of-day and evening peak periods. It is seen that the CDF of travel time along the path in Figure 6.6(a) for three peak periods dominates other(s), only when the on-time arrival probability is low. However, it is dominated by other(s) when the on-time arrival probability becomes higher. It reflects the risk of this path.

6.1.2 From the southwest suburbs to Chicago downtown

The above experiments show that a detour on I-55 is even more reliable than using I-290 for the morning peak period (see Figure 6.6(c)). It implies that I-55 is less congested than I-290. I-55 is a popular route connecting the southwest suburbs with Chicago downtown. Experiments are also conducted to test the reliability of I-55. The intersection of Michigan Ave. and Randolph Dr. is still selected to represent the downtown, and the intersection of E. 67th St. and S. Cicero Ave. is selected to represent the southwest suburbs.

Google and Yahoo Maps suggest motorists using S. Cicero Ave, and then entering I-55 North at Exit 286, and finally using Lake Shore Dr to downtown, as shown in Figure 6.8(a). Experiments show that this path is always shortest for the mid-of-day and evening peak periods for any on-time arrival probability. For the morning peak period, the shortest path is only different in the downtown area, as shown in the left top corner in Figure 6.8(a): motorists are suggested to leave I-55 North at Exit 293A and then to drive along S. Clark St to the destination.

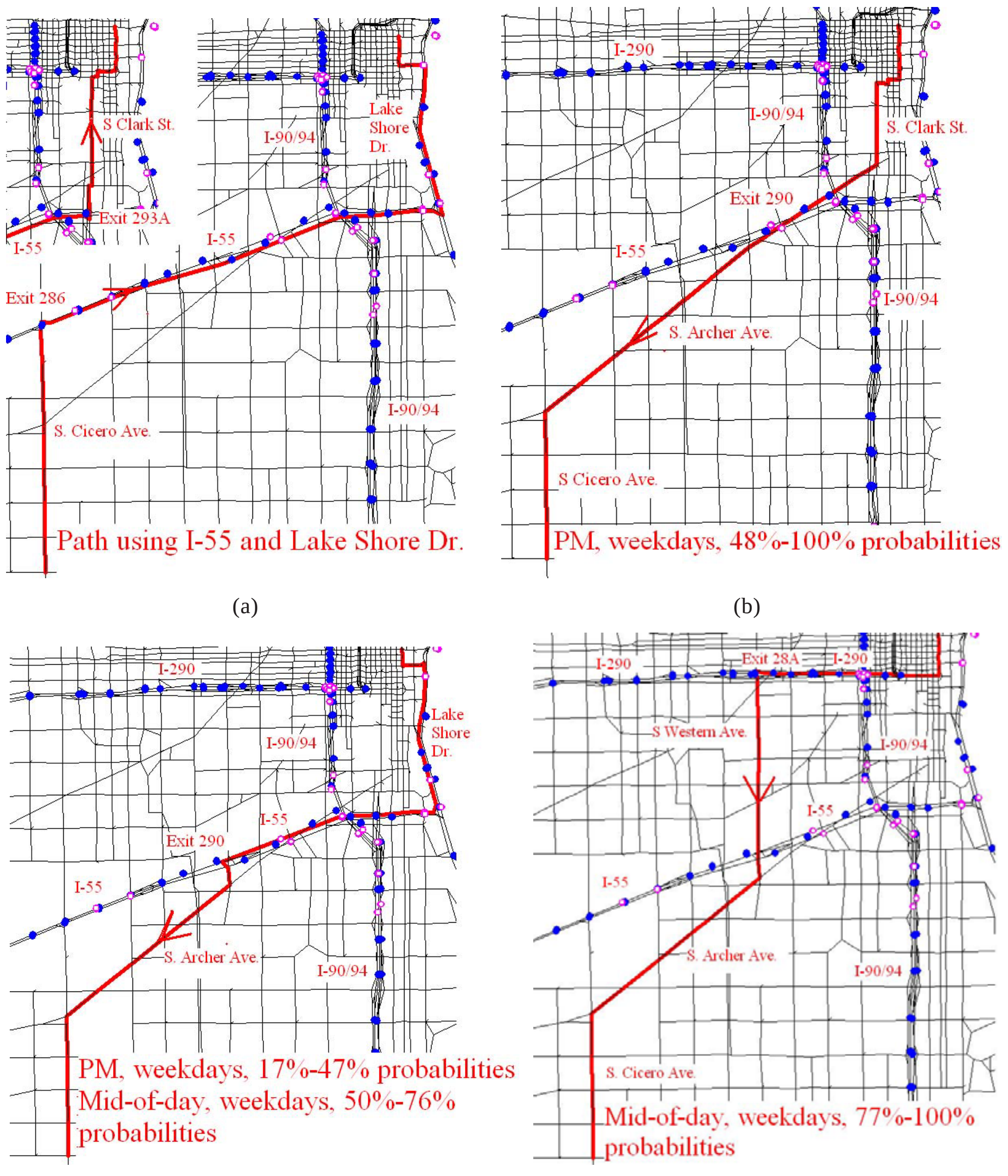


Figure 6.8: Shortest paths from from southwest suburbs to Chicago downtown or in the reverse direction

If driving in the reverse direction, i.e, from Chicago downtown to the southwest suburbs, the typical path suggested by Google and Yahoo Maps is still the same to the path shown in Figure 6.8(a) but in the reverse direction. The reliable route guidance shows that it is always shortest for the morning peak period, for any on-time arrival probability. However, this path becomes risky for the mid-of-day and evening peak periods. It is shortest only when the on-time arrival probability is no higher than 49% (mid-of-day) or 17% (evening peak). For these two periods, motorists are suggested to avoid using I-55 entirely if they prefer a high on-time arrival probability: $\geq$ 77% for mid-of-day period, and $\geq$ 48% for the evening peak period, as shown in Figure 6.8(d) and 6.8(b). For example, according to the reliable route guidance, motorists should use I-290 West first, then use arterial streets (S. Western Ave., S Archer Ave.) before they reach S. Cicero Ave (see Figure 6.8(d)), for the mid-of-day period. For 95% on-time arrival probability, this path requires a time budget of only 25 minutes and 2 seconds, while the path suggested by Google and Yahoo maps (see in Figure 6.8(a), the reverse direction) requires 30 minutes and 35 seconds. This routing policy leads a 21% saving in travel time. For the evening peak period, motorist are suggested using arterial streets all the time, as shown in Figure 6.8(b), and the required travel time is 27 minutes and 8 seconds. Compared with the path in Figure 6.8(a) (the reverse direction) that needs 36 minutes and 4 seconds, the routing policy leads a 33% saving of travel time! Therefore, our experiments indicate that I-55 South is more congested for the mid-of-day and evening periods than for the morning peak period. On the other hand, I-55 North is always reliable all the time.

Figure 6.1.2 reports the CDFs of travel time on the FSD-admissible paths for trips from Chicago downtown to the southwest suburbs for three periods. The figures on the left and right demonstrate well the reason why using the paths in Figure 6.8(b) and 6.8(d) can save a lot of time, compared with the path in Figure 6.8(a) (the reverse direction), for a 95% on-time arrival probability. However, for the morning peak periods, there is only one FSD-admissible path, as shown in the middle one.

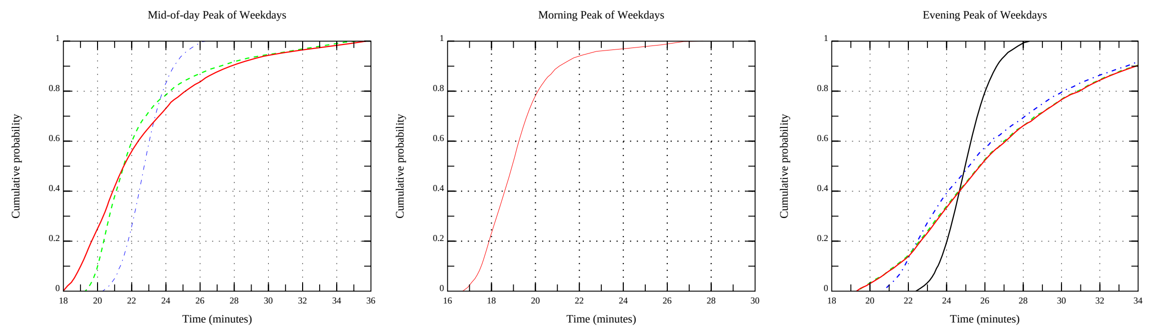


Figure 6.9: Cumulative density functions (CDFs) of travel time on the FSD-admissible paths for trips from Chicago downtown to the southwest suburbs for three periods

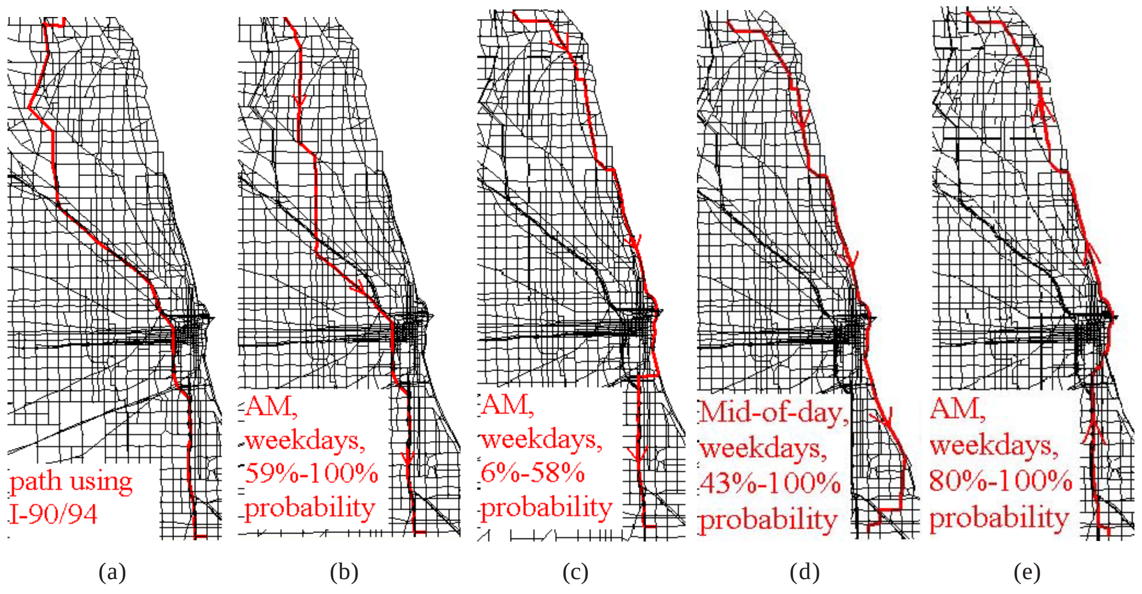

path using I-90/94
AM, weekdays, 59%-100% probability
AM, weekdays, 6%-58% probability
Mid-of-day, weekdays, 43%-100% probability
AM, weekdays, 80%-100% probability
(a)
(b)
(c)
(d)
(e)

Figure 6.10: Shortest paths from northshore to south suburbs or in the reverse direction

6.2 From the northshore to south suburbs

The challenge of this routing problem (see Figure 5.1) is how to travel through the Chicago downtown and its congested peripheral area. Typically, motorists have two options: I-90/94 and Lake Shore Dr. The origin and the destination are deliberately selected so that both options could become attractive. Specifically, the intersection of Illinois Rd. and Locust Rd. on the northshore, and the intersection of E.79 St. and S Martin Luther King Dr. in the south suburbs are selected. The reliable route guidance generally suggests that for weekdays motorists should stay away from I-90/94, especially for the portion north of Chicago downtown if driving from north to south. Our results indicate that the popular choice shown in Figure 6.10(a) is shortest only when the on-time arrival probability is very low (6% for morning peak, and 18% for mid-of-day). For the evening peak, this path is not even FSD-admissible. For the mid-of-day and the evening peak periods, Lake Shore Dr. is more reliable. Figure 6.10(d) shows that the shortest path during the mid-of-day period uses the Lake Shore Dr, which guarantees an on-time arrival probability higher than 43%. The FSD-admissible paths for the evening peak are similar and not reported separately. For the morning peak, however, Lake Shore Dr. is preferred only if a motorist wants to arrive on time with a probability lower than or equal to 59% (see Figure 6.10(c)). For higher reliability motorists need to use various arterial streets until they are close to downtown, and then switch I-90/94 (see Figures 6.10(b)).

Driving from south to north during weekdays is a different story. Experiments show that the path in Figure 6.10(a) (reverse direction) represents most of FSD-admissible path with the exception of the morning peak. For that period, Lake Shore Dr. is always recommended and any path using I-90/94 is not FSD-admissible.

6.3 Computational performance

Table 6.1 reports the consumed CPU times for solving each of the four routing problems in either direction and in both weekends and weekdays [1]. As shown, the RASP problem was solved within 30 seconds in most cases. Considering the complexity of the problem and the sheer size of the network, the performance is acceptable even from a practical point of view. Note that our algorithm actually finds FSD-admissible paths from all origins to the destination simultaneously. Therefore, once the computation is done for one O-D pair, little further efforts are needed to obtain admissible paths from another origin to the same destination. It may be possible to expedite the computation if the route guidance is only needed for one O-D pair. This possibility, however, is not further explored in the current implementation.

Table 6.1 reveals that solving the weekend models generally took shorter CPU times and generated fewer admissible paths. We conjecture that this is because the roads are less congested on weekends, and therefore subject to smaller travel time variances. Par-

[1]In fact, the same travel time distributions are used on arterial streets for both weekends and weekdays. Covered links, however, have different distributions on weekends and weekdays from observations.

Table 6.1: Computation performance of the algorithm in the two routing problems

	Weekdays			Weekends		
	AM Peak	Mid-of-day	PM Peak	AM Peak	Mid-of-day	PM Peak
Downtown to ORD						
CPU time	29.58	18.69	16.58	12.25	19.14	8.50
# paths	7	5	4	1	5	1
ORD to downtown						
CPU time	29.58	23.70	14.58	15.69	15.36	28.02
# paths	6	2	2	1	2	4
West suburbs to downtown						
CPU time	108.03	29.30	26.66	15.86	30.23	15.42
# paths	5	3	2	1	3	4
Downtown to west suburbs						
CPU time	55.72	21.84	32.39	13.27	11.22	9.42
# paths	13	3	11	2	2	1
Southwest suburbs to downtown						
CPU time	105.41	27.45	27.28	15.81	19.33	15.52
# paths	1	1	1	1	2	1
Downtown to southwest suburbs						
CPU time	71.25	66.05	88.17	14.39	33.59	21.09
# paths	1	3	4	1	1	1
Northshore to south suburbs						
CPU time	65.88	74.39	20.42	15.52	46.53	33.74
# paths	7	10	2	2	1	4
South suburbs to northshore						
CPU time	60.83	39.00	33.74	14.19	36.25	12.08
# paths	10	6	6	1	3	1

Note: 1) CPU time is measured in seconds; 2) ”# paths” stands for the number of FSD-admissible paths

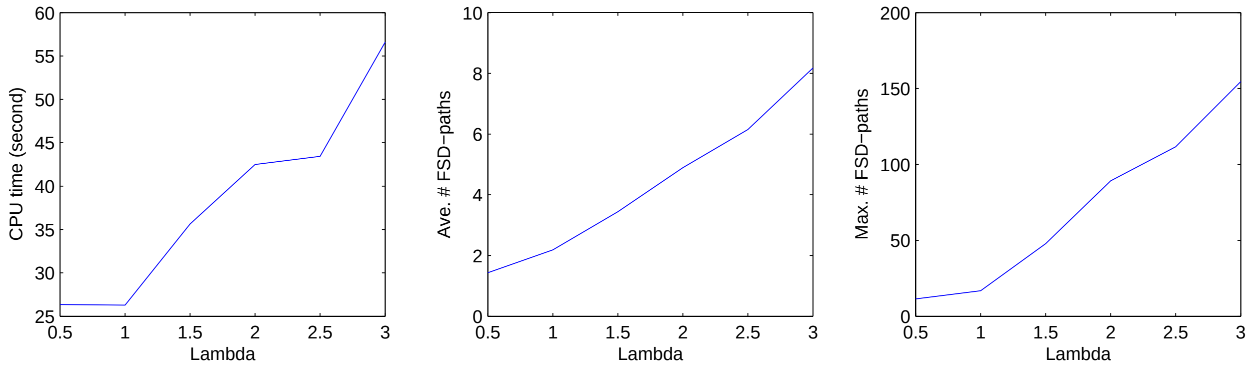


Figure 6.11: Impacts of variances on arterial streets on computational performance.

ticularly, the morning and evening periods on weekends often have only one admissible path; the most obvious choice (i.e. major freeways) usually prevails in those cases.

In order to better understand the impacts of travel time variances, a sensitivity analysis is conducted in the following. The focus is given to the variances on arterial streets, since the linear regression model used to estimate them was not fitted very well. In the analysis, we simply multiply the estimated variances on arterial by a parameter $\lambda = 0.5, 1.0, 1.5, 2.0, 2.5,$ and 3.0. For each value of λ (called a scenario), five different locations in the CMAP network are selected to compute the all-to-one reliable shortest paths. Three performance indexes are recorded for each run: the CPU time, the average and maximum number of FSD-admissible paths at all nodes. The average of the five runs (each for one destination) are used as the average index for the scenario. Figure 6.11 shows how these average indexes vary with the value of λ. Clearly, as the variances increase, the average size of FSD-admissible paths grows. As a consequence, it takes the algorithm longer to solve the problem; note that the complexity of the algorithm depends on the size of FSD-admissible paths (Nie & Wu 2009*b*). The good news is that the CPU time does not seem to increase superlinearly with the variances. In fact, the CPU time barely doubled when the variances on arterial streets become six time higher. The longest average CPU times is still less than one minute, which remains acceptable for practical purposes.

6.4 Summary

To summarize, our experiments indicate that best paths do vary substantially with the reliability requirement, measured in this report by the probability of arriving on-time or earlier. For motorists who travel during rush hours and seek high reliability, reliable route guidance could generate up to 10 - 30 % of travel time savings. Interestingly, highly reliable routes often tend to prefer major arterial to freeways and highways in rush hours. This phenomenon could well have been caused by the underestimation of travel time variances on arterial streets, recalling that the distributions on arterial streets were estimated using linear models calibrated from freeway data, which were not fitted very well for the variances. Nevertheless, staying away from congested freeways during rush hours, particularly when you have important appointments, does agree with conventional wisdom. Such advice appeals to many motorists probably because freeways are less amenable to effective recourses when uncertainty strikes. The results from our case study may have provided a piece of empirical evidence to support this perception.

Chapter 7

Conclusions

Travel time reliability is important to highway users. Personal and business travelers value reliability because it allows them to make better use of their own time. Shippers and freight carriers require predictable travel times to remain competitive. The lack of travel reliability forces motorists to choose between running the risk of being late (therefore missing important appointments or just-in-time deliveries) or budgeting a large buffer time, much of which is often wasted. In order to hedge against such uncertainty, highway users badly need decision-supporting tools that are capable of exploiting existing data sources to 1) reveal and document random pattern of travel times of highway networks and 2) provide real-time route guidance that takes uncertainty into consideration. This project confronts the above challenge by implementing a reliable a prior shortest path (RASP) algorithm in a software tool named Chicago Travel Reliability (CTR), and test it using real traffic data from the Chicago area. We first summarize what have been accomplished in this project, and then discuss possible direction for further research.

7.1 Main results

First, GIS and traffic database in the Gary-Chicago-Milwaukee (GCM) traveler information system were processed and analyzed. GIS data from the GCM system were first merged with another data set obtained from Chicago Metropolitan Agency for Planning (CMAP), which provides important static road properties missed from GCM data (such as capacity and free flow speed). Tools were developed to automate the merge of the two

GIS data sources, which involves coordinate conversion and position matching. GCM traffic data include two parts: loop detector data (speed, volume and occupancy), and IPASS transponder data (travel time between toll booths). In the Chicago area, the system has 827 loop detectors and 309 IPASS OD pairs and has operated since October of 2004. In total, there are approximately 500 million records in the traffic database. A Windows console application named GCM-Gateway was developed to download these traffic data and convert them (stored in Postgresql format) into a MYSQL database on a workstation at Northwestern University.

Second, a RASP algorithm was implemented using Visual C++ and tested on the Chicago network. Our experiments indicate that best paths do vary substantially with the reliability requirement, measured in this report by the probability of arriving on-time or earlier. For motorists who travel during rush hours and seek high reliability, reliable route guidance could generate up to 10 - 30 % of travel time savings comparing to the conventional routing mode. This significant benefit could well justify the extra efforts to generate reliable guidance. The study also verifies the capability of existing algorithms in solving large-scale reliable routing problems within reasonable amount of time. As noted, in most cases the algorithm found all-to-one reliable paths (FSD-admissible paths) within a minute on an up-to-date workstation. Considering the non-deterministic polynomial complexity of the problem and the sheer size of the network, the reported performance is deemed satisfactory.

Thirdly, a software tool named Chicago Travel Reliability (CTR) was developed, which provides an integrated environment to: 1) visualize and analyze traffic data, 2) construct and display travel reliability measures for the Chicago network, and 3) provide reliable route guidance and compare it with conventional routing algorithms. CTR is currently available upon email request (y-nie@norhtwestern.edu) and will be posted on-line soon.

7.2 Future Work

As the first phase of a contiual effort, this project is focused mainly on the demonstration of reliable routing through the dissemination of CTR. It is the first of our four-step implementation strategy which consists of demonstration, initial deployment, market analysis and response and full deployment (commercialization). Presently, CTR has a number of technical limitations that restrict its ability to sufficiently support further deployment.

First of all, the current project has focused on GCM system whose loop detector and I-PASS transponder data cover only interstate highways and expressways. Since these links constitute only a small portion of all network links, they might be inadequate for accurate route guidance. The method used to deal with the lack of arterial data (detailed in Chapter 5) is a compromise rather than a resolution. In order to obtain travel time data on arterial streets, additional data sources should be exploited. One possibility that the research team is exploring is to use automatic vehicle location (AVL) data archived by Chicago Transit Authority (CTA). Since CTA bus routes cover most strategic arterial streets in the City of Chicago and its neighboring suburban areas, the AVL data from CTA holds promise in filling an important gap in GCM data currently used by CTR.

Secondly, to perform reliable route guidance in real-time (e.g., through an en-vehicle navigation system), further improvements are still needed. Possible strategies include but are not limited to: imposing higher-order stochastic dominance to reduce the number of non-dominant paths; adopting approximation methods; and exploiting the special properties of a one-to-one (instead of all-to-one) shortest path problem.

Finally, the current interface of CTR should be further enhanced. For example a simpler and web-based version of CTR could attract more users and make it easier to collect user feedback about the software.

The above limitations will be addressed in the next phase, which is currently funded by CCITT for the year of 2009 - 2010.

Bibliography

Andreatta, G. & Romeo, L. (1988), 'Stochastic shortest paths with recourse', *Networks* **18**(3), 193–204.

Bander, J. L. & White, C. C. (2002), 'A heuristic search approach for a nonstationary stochastic shortest path problem with terminal cost', *Transportation Science* **36**(2), 218–230.

Bard, J. & Bennett, J. (1991), 'Arc reduction and path preference in stochastic acyclic networks', *Management Science* **37(2)**, 198–215.

Brumbaugh-Smith, J. & Shier, D. (1989), 'An empirical investigation of some bicriterion shortest path algorithms', *European Journal of Operational Research* **43**(2), 216–224.

Carraway, R. L., Morin, T. L. & Moskowitz, H. (1990), 'Generalized dynamic programming for multicriteria optimization', *European Journal of Operational Research* **44**(1), 95–104.

Chen, Y., Bell, M. G. H., Wang, D. & Bogenberger, K. (2006), 'Risk-averse time-dependent route guidance by constrained dynamic a* search in decentralized system architecture', *Transportation Research Record* **1944**, 51–57.

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

CMAP (2006), Travel demand modeling for the conformity process in northeastern illinois, available at http://www.cmap.illinois.gov /uploaded-files/publications/other_publications /pm25_conformity_analysis_b.pdf, last accessed on 6/10/2009, Chicago Metropolitan Agency for Planning.

Croucher, J. (1978), ‘A note on the stochastic shortest route problem’, *Naval Research Logistics* **25**(4), 729–732.

Eiger, A., Mirchandani, P. B. & Soroush, H. (1985), ‘Path preferences and optimal paths in probabilistic networks’, *Transportation Science* **19**(1), 75–84.

Fan, Y., Kalaba, R. & Moore, J. (2005*a*), ‘Arriving on time’, *Journal of Optimization Theory and Applications* **127**(3), 497–513.

Fan, Y., Kalaba, R. & Moore, J. (2005*b*), ‘Shortest paths in stochastic networks with correlated link costs’, *Computers and Mathematics with Applications* **49**(9-10), 1549–1564.

Fan, Y. & Nie, Y. (2006), ‘Optimal routing for maximizing the travel time reliability’, *Networks and Spatial Economics* **3**(6), 333–344.

Frank, H. (1969), ‘Shortest paths in probabilistic graphs’, *Operations Research* **17**(4), 583–599.

Fu, L. (2001), ‘An adaptive routing algorithm for in-vehicle route guidance systems with real-time information’, *Transportation Research Part B* **35**(8), 749–765.

Fu, L. & Rilett, L. R. (1998), ‘Expected shortest paths in dynamic and stochastic traffic networks’, *Transportation Research Part B* **32**(7), 499–516.

Gao, S. & Chabini, I. (2006), ‘Optimal routing policy problems in stochastic time-dependent networks’, *Transportation Research Part B* **40**(2), 93–122.

Hall, R. W. (1986), ‘The fastest path through a network with random time-dependent travel time’, *Transportation Science* **20**(3), 182–188.

Hansen, P. (1979), Bicriterion path problems, *in* G. Fandel & T. Gal, eds, ‘Multiple criteria decision making theory and application’, pp. 109–127.

Henig, M. I. (1985), ‘The shortest path problem with two objective functions’, *European Journal of Operational Research* **25**(2), 281–291.

Kaparias, I., Bell, M., Chen, Y. & Bogenberger, K. (2007), ‘Icnavs: a tool for reliable dynamic route guidance’, *IET Intellegent Transportation Systems* **1**(4), 225–253.

Levy, H. & Hanoch, G. (1970), ‘Relative effectiveness of efficiency criteria for portfolio selection’, *Journal of Financial and Quantitative Analysis* **5**, 63–76.

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

Miller-Hooks, E. (1997), Optimal Routing in Time-Varying, Stochastic Networks: Algorithms and Implementations, PhD thesis, Department of Civil Engineering, University of Texas at Austin.

Miller-Hooks, E. D. (2001), ‘Adaptive least-expected time paths in stochastic, time-varying transportation and data networks’, *Networks* **37**(1), 35–52.

Miller-Hooks, E. D. & Mahmassani, H. S. (1998), ‘Least possible time paths in stochastic, time-varying networks’, *Computers and Operations Research* **25**(2), 1107–1125.

Miller-Hooks, E. D. & Mahmassani, H. S. (2000), ‘Least expected time paths in stochastic, time-varying transportation networks’, *Transportation Science* **34**(2), 198–215.

Miller-Hooks, E. D. & Mahmassani, H. S. (2003), ‘Path comparisons for a priori and time-adaptive decisions in stochastic, time-varying networks’, *European Journal of Operational Research* **146**(2), 67–82.

Mirchandani, P. B. (1976), ‘Shortest distance and reliability of probabilistic networks’, *Computers and Operations Research* **3**(4), 347–355.

Montemanni, R. & Gambardella, L. (2004), ‘An exact algorithm for the robust shortest path problem with interval data’, *Computers and Operations Research* **31**(10), 1667–1680.

Murthy, I. & Sarkar, S. (1996), ‘A relaxation-based pruning technique for a class of stochastic shortest path problems’, *Transportation Science* **30**(3), 220–236.

Murthy, I. & Sarkar, S. (1998), ‘Stochastic shortest path problems with piecewise linear concave linear functions’, *Management Science* **44**(11), 125–136.

Nie, Y. (2006), *A Programmer's Manual for Models and Algorithms Toolkit (MAT)*, University of California, Davis.

Nie, Y. & Fan, Y. (2006), 'Arriving-on-time problem: Discrete algorithm that ensures convergence', *Transporttion Research Record* **1964**, 193–200.

Nie, Y. & Wu, X. (2009*a*), 'Reliable a priori shortest path problem with limited spatial and temporal dependencies', *In Proceedings of the 18th International Symposium on Transportation and Traffic Theory, accepted* .

Nie, Y. & Wu, X. (2009*b*), 'Shortest path problem considering on-time arrival probability', *Transportation Research Part B* **43**, 597–613.

Polus, A. (1979), 'A study of travel time and reliability on arterial routes', *Transportation* **8**(2), 141–151.

Polychronopoulos, G. H. & Tsitsiklis, J. N. (1996), 'Stochastic shortest path problems with recourse', *Networks* **27**(2), 133–143.

Provan, J. S. (2003), 'A polynomial-time algorithm to find shortest paths with recourse', *Networks* **41**(2), 115–125.

Schrank, D. & Lomax, T. (2007), 2007 urban mobility study, Technical report, Texas Transportation Institute, Texas A&M University, College Station, TX.

Sen, S., Pillai, R., Joshi, S. & Rathi, A. (2001), 'A mean-variance model for route guidance in advanced traveler information systems', *Transportation Science* **35**(1), 37–49.

Sigal, C. E., Alan, A., Pritsker, B. & Solberg, J. J. (1980), 'The stochastic shortest route problem', *Operations Research* **28**(5), 1122–1129.

Sivakumar, R. & Batta, R. (1994), 'The variance-constrained shortest path problem', *Transportation Science* **28**(4), 309–316.

von Neumann, J. & Morgenstern, O. (1967), *Theory of Games and Economic Behavior*, Wiley, New York.

Waller, S. T. & Ziliaskopoulos, A. K. (2002), ‘On the online shortest path problem with limited arc cost dependencies’, *Networks* **40**(4), 216–227.

Wu, X. & Nie, Y. (2009), ‘Implementation issues in approximate algorithms for reliable a priori shortest path problem’, *Journal of the Transportation Research Board* **Forthcoming**.

Yu, G. & Yang, J. (1998), ‘On the robust shortest path problem’, *Computers and Operations Research* **25**(6), 457–468.

Appendix A

Class implementation

Implementation details of new and affected classes are reported here. Note that class members and methods that are not related to the routing problem are mostly ignored.

CLASS TNM_SNET   PARENT CLASS: N/A

Data Members:

string networkName	– name of the network.
int numOfNode	– number of nodes in the network.
int numOfLink	– number of links in the network.
int numOfOrigin	– number of origin pairs.
int numOfOD	– number of OD pairs.
vector<TNM_SLINK*> linkVector	– a vector storing link pointers.
vector<TNM_SNODE*> nodeVector	– a vector storing node pointers.
vector<TNM_SORIGI*N> originVector	– a vector storing origin pointers.
vector<TNM_SNODE*> destNodeVector	– a vector storing all destination.
SCANLIST* scanList	– a scanlist object which determines how shortest path algorithm is implemented.
int buildStatus	– = 0 means network has not been build yet.
int initialStatus	– = 0 means network has not been initiated yet.

Constructor:

void TNM_SNET(const string& netName) – a constructor, given the network name.

Return	–	none.

Operations:

int BuildFort(const TNM_LINKTYPE tt) – load network from specific documents

Return	–	0 if loading the network succeeds; a non-zero number otherwise.

int CheckBuildStatus(bool noteBuilt) – check the network is built or not

Return	–	= 0 means network has not been build yet.

TNM_SLINK* AllocateNewLink(const TNM_LINKTYPE &pType) – Allocate memory to a new link according to the link type

Return	–	a TNM_SLINK pointer

TNM_SNODE* AllocateNewNode(const TNM_NODETYPE &nType) – Allocate memory to a new node according to the node type

Return	–	a TNM_SNODE pointer

CLASS TNM_DNET PARENT CLASS: TNM_SNET

Data Members:

RoutingType routingType	–	fixed, mixed, variable default = fixed.
smallInt RTUpdateFreq	–	the update frequency of routing table.
largeInt assignHorizon	–	the largest assignment horizon through all O-D pairs.
int numOfKSP	–	when generating an initial assignment scheme, the number of K-shortest path.
floatType m_demandLevel	–	indicate the demand level.
static smallInt unitTime	–	unit assignment time interval.
bool m_releaseVehPerLoad	–	release vehicle per loading interval.
static smallInt flowScalar	–	how many ”vehicles” is needed to represent one unit flow.

Constructor:

void TNM_DNET(const string& netName) – a constructor, given the network name.

Return	–	none.

CLASS TNM_PROBNET   PARENT CLASS: TNM_DNET

Data Members:

int numOfInterval	– number of interval used for discretization scheme.
bool m_saveMemoryMode	– false: using the vector that helps track paths, improving efficiency but using more memory; otherwise true.
int m_scannedNode	– the number of nodes that are already scanned.
int m_tobeScannedNode	– the number of nodes that are to be scanned.

Constructor:

void TNM_ProbNet(const string& netName) – a constructor, given the network name.

Return – none.

Operations:

int BuildFortProb(const TNM_LINKTYPE tt, const TNM_NODETYPE nt) – load stochastic network, given the link and node types.

Return – 0 if loading the network succeeds; a non-zero number otherwise.

int InitializeLinkProbability(int indicator) – initialize the distributions of link travel times, given indicator 0 : AM peak hour, 1 : PM peak hour, 2 : middle of the day, 3 : offpeak time

Return – 0 if initializing succeeds; a non-zero number otherwise.

bool InitiPath(int sd, TNM_PNODE* des) – initialize the path in the whole network.

Return – 0 if initializing succeeds;; a non-zero number otherwise.

TNM_PNODE* ReadInDestination() – read the destination

Return – the pointer of the destination node

floatType SolveNonDominancePath(TNM_PNODE* des, int sd, bool record, bool EDA) – solve all FSD-admissible paths based on a destination node, given specific solution conditions

Return – the cpu time.

floatType* GenGammaDistribution(floatType mean, floatType variance, floatType mu, floatType &minimum, floatType &maximum, CMatlabEng *matlab) – generate a Gamma distribution for link travel time, based on given parameters

Return – an array containing supporting points.

void WriteDistribution(CMatlabEng *matlab, int interval = 100) – write distribution into a specific file

Return – none

void GetLinkDistributionFromDisFile(TNM_ProbDist &pDist, int stpID, int indicator= 0) – get the distribution of a given link and a given time period from files

Return – none.

vector<TNM_ProbPath $*$ > GetPathAtNode(int NodeID) – get all FSD-admissible path at a specific node, given the node id

Return – a vector containing FSD-admissible paths.

vector<TNM_ProbPath $*$ > GetPathAtNode(TNM_PNODE *node) – get all FSD-admissible path at a specific node, given the node pointer

Return – a vector containing FSD-admissible paths.

TNM_PPATH* GetOptimalPath(int NodeID, floatType prob) – get the optimal path at a specific node and a specific on-time arrival probability, given a node id

Return – the optimal path pointer.

TNM_PPATH* GetOptimalPath(TNM_PNODE *node, floatType prob) – get the optimal path at a specific node and a specific on-time arrival probability, given a node pointer

Return – the optimal path pointer.

int WriteDisFromBinaryDisFile(int indicator) – transfer the distribution in binary document to text document

Return – 0 if succeeds; a non-zero number otherwise.

int PrintPerformance(double duration) – print out the performance to files

Return – 0 if succeeds; a non-zero number otherwise.

void PrintOptimalAOTPolicy(floatType availableTime) – print out the optimal path based on the given travel time budget

Return – none.

Overridables:

TNM_SNODE* AllocateNewNode(const TNM_NODETYPE &ntype) – Allocate memory to a new node according to the given node type

Return – a node pointer

TNM_SLINK* AllocateNewLINK(const TNM_LINKTYPE &pType) – Allocate memory to a new link according to the given link type

Return – a link pointer

CLASS TNM_SNODE PARENT CLASS: N/A

Data Members:

int id	– the node ID.
TNM_NODETYPE type	– node type.
largeInt xCord	– x coordinate of the node.
largeInt yCord	– y coordinate of the node.
vector<TNM_SLINK∗ > forwStar	– a vector containing the IDs of the links that go out from this node.
vector<TNM_SLINK∗ > backStar	– a vector containing the IDs of the links that point this node.
PATHELEM* pathElem	– store the optimal routing policy, and normally this is reserved for shortest path tree rooted at an origin.
PATHELEM* rPathElem	– this is reserved for possible all-to-one shortest path search.
int attachedOrg	– how many origins are built on this node.
bool m_isThrough	– whether this node is traversed by a path or not.

Constructor:

void TNM_SNODE() – a default constructor

Return – none.

Operations:

bool Initialize(const NODE_VALCONTAINER &cont) – virtual function, used for node initialization

Return	– true

inline int id_() – get the node id

Return	– node id.

bool IsControlled() – whether this node is controlled or not

Return	– true if controlled; false otherwise.

TNM_SLINK* DestDummyAttached() – whether or not its outgoing link contains a dummy destination

Return	– the link pointer if the outgoing link contains a dummy destination; null otherwise.

TNM_SLINK* OrigDummyAttached() – whether or not its incoming link contains a dummy origin

Return	– the link pointer if the incoming link contains a dummy origin; null otherwise.

Class TNM_DNODE   Parent class: TNM_SNODE

Data Members:

TNM_ROUTINGTAB routingTable	– give the adaptive optimal routing policy.
PATHELEM** tdRouting	– give the optimal routing policy, used for time-dependent routing.
NETSIMPATHELEM** tdSPTree	– store the information of the next link on the spanning tree.

Constructor:

void TNM_DNODE() – a default constructor.

Return	– none.

Class TNM_PNODE   Parent class: TNM_DNODE

Data Members:

vector<TNM_PPATH∗ > m_stochpath	– store the FSD-admissible paths.

PATHELEM** tdRouting	– give the optimal routing policy, used for time-dependent routing.
bool m_isDestination	– whether or not this node is the destination.
int m_nInterval	– the number of discrete points describing the distributions.

Constructor:

void TNM_PNODE() – a default constructor

Return – none.

Operations:

void UpdateFrontier(int SD) – update the frontier

Return – none

TNM_PPATH* CreateNewPath(TNM_PRBLK *lk, TNM_PPATH* currentPath, int SD, bool record, fstream *outfile) – create a new path

Return – the pointer of the new path.

TNM_PRBLK* GetLink(TNM_PNODE *to) – get the link betwenn two given nodes

Return – link pointer if the link exists; null otherwise.

bool CheckNewPathWithFrontier(TNM_PPATH *newpath, int SD, bool EDA, bool record, fstream *outfile1, bool memorySave) – check whether the given new path can dominate the current frontier or not

Return – true if dominates; false otherwise.

void ClearPath(bool record, fstream *outfile1) – delete all FSD-admissible path and clear the memory

Return – none.

bool NodeUpdate(TNM_PNODE *to, int SD, bool EDA, bool record, fstream *outfile1, bool memorySave) – check whether the set of FSD-admissible paths at current node can be updated or not based on the FSD-admissible paths on the given downstream node

Return – true if the set of FSD-admissible paths is updated; false otherwise.

bool PathComparison(TNM_PPATH *newpath, int SD, bool record, fstream *outfile1, bool memorySave) – check whether the given new path can dominate (or be dominated by) the current FSD-admissible paths or not at this node

Return	– true if the new path dominates any existing FSD-admissible paths; false otherwise.

int PrintPathAtNODE(vector<TNM_ProbPath∗ > solutionPaths, int sd, string &networkName) – print out the distribution of all FSD-admissible paths at the current node

Return	– 0 if succeeds; a non-zero number otherwise.

bool CheckRepeatPath(TNM_PPATH *thatpath, bool record, fstream *outfile1) – check whether or not the given path already exists in the set of the current FSD-admissible paths

Return	– true if repeat path appears; false otherwise.

void ClearNode() – clear all memory associated with the current node

Return	– none.

void SetNumberOfInterval(int n) – set the number of discrete points

Return	– none.

int GetNumberOfInterval() – get the number of discrete points

Return	– number of discrete points.

Class TNM_SLINK   Parent class: N/A

Data Members:

int id	– the link ID.
int orderID	– the order ID of the link in the vector.
TNM_LINKTYPE type	– link type.
TNM_SNODE *head	– the pointer of the node that the link points to.
TNM_SNODE *tail	– the pointer of the node where the link goes out.
floatType capacity	– the capacity of the link.
floatType volume	– the volume of the link.
floatType length	– the length of the link.
floatType ffs	– the free flow speed of the link.
floatType fft	– the free flow travel time of the link.

floatType cost	– the general cost of the link.
floatType fdCost	– the derivative of link cost.
floatType* buffer	– it is the working space for outside using.
floatType markStatus	– a status variable for temporary usage.
TNM_SLINK *revLink	– the pointer toward the link on the reverse direction.
bool dummy	– if the link is a temporary ”dummy” link.
vector<TNM_SPATH∗ > pathInciPtr	– a vector contains all path pointers which use the link.

Constructor:

void TNM_SLINK() – a default constructor

Return	– none.

Operations:

bool Initialize(const LINK_VALCONTAINER cont) – initialize the link

Return	– true if succeeds; false otherwise.

void ConnectFW() – put the link’s pointer to tail node’s forwStar

Return	– none.

void ConnectBK() – put the link’s pointer to head node’s backStar

Return	– none

void DisconnectFW() – erase link’s pointer from tail’node’s forwStar

Return	– none

void DisconnectBK() – erase link’s pointer to head node’s backstar

Return	– none

bool CheckParallel() – check whether or not two links are parallel

Return	– true if parallel; false otherwise.

Class TNM_DLINK   Parent class: TNM_BPRLK

Data Members:

floatType laneHldCap	– hold capacity per lane: veh / mile / lane.
floatType laneFlwCap	– flow capacity per lane: veh / hr / lane.

tinyInt numOfLanes	– the number of lanes.
smallInt inVehs	– number of vehicles entering into the link at current time interval.
smallInt unitFFT	– free flow travel time in unit loading time.
short dirIndex	– the direction index of the outgoing link with respect a node.
short bDirIndex	– backward direction index.
largeInt cumIn	– it is used when output cumulative file.
largeInt cumOut	– it is used when output.
largeInt* tdTT	– travel time array, will be allocated and deleted as required.
largeInt* tdCumIn	– time-dependent cumin.
largeInt* tdCumOut	– time-dependent cumout file.
floatType* tdCost	– this is used for arrival on time, store the probability of each possible travel time.

`Constructor:`

void TNM_DLINK() – a default constructor

Return – none.

`Overridables:`

bool Initialize(const LINK_VALCONTAINER cont) – initialize the link

Return – true if succeeds; false otherwise.

void ConnectFW() – put the link's pointer to tail node's forwStar

Return – none.

void ConnectBK() – put the link's pointer to head node's backStar

Return – none

void DisconnectFW() – erase link's pointer from tail'node's forwStar

Return – none

void DisconnectBK() – erase link's pointer to head node's backstar

Return – none

Class `TNM_PBRLINK` Parent class: `TNM_DLINK`

`Data Members:`

int m_stampid – the stamp ID of the link.

TNM_ProbDist* m_pdist – probability distribution.

Constructor:

void TNM_PBRLINK() – a default constructor

Return – none.

Operations:

bool CreateDist() – create the distribution of link travel time

Return – true if succeeds; false otherwise.

void DeleteDist() – erase the distribution and release the memory

Return – none.

TNM_ProbDist* GetDist() – get the pointer of the distribution

Return – the pointer of the distribution.

string GetStampID() – convert the stamp ID to string

Return – the string of stamp ID

Overridables:

void ConnectFW() – put the link's pointer to tail node's forwStar

Return – none.

void ConnectBK() – put the link's pointer to head node's backStar

Return – none

void DisconnectFW() – erase link's pointer from tail'node's forwStar

Return – none

void DisconnectBK() – erase link's pointer to head node's backstar

Return – none

Class TNM_GCMLINK Parent class: TNM_PBRLINK

Data Members:

enum DTYPE DT_NONE,
DT_DETECTOR, – type of the detectors.
DT_IPASS, DT_DETIPASS

string m_roadName – store road name.

string m_dirType	– direction type.
vector<TNM_PDetector∗ > m_pdetVec	– the point detectors that use the link.
vector<TNM_LDetector∗ > m_ldetVec	– the line detector that use this link.
vector<double> m_ldetRatio	– how much weight this ldet carries for this link.
DTYPE m_detType	– the type of detectors.
bool m_dbDataEffect	– the data collected from database is effective or not, for example, some link has data such min¿max, so this is ineffective data.

Constructor:

void TNM_GCMLINK() – a default constructor

Return – none.

Overridables:

bool Initialize(const LINK_VALCONTAINER cont) – initialize the link

Return – true if succeeds; false otherwise.

void Print() – print out the result

Return – none.

Class TNM_SPATH Parent class: N/A

Data Members:

int id	– the path ID.
floatType* stochTD	– store the third order stochastic data.
floatType flow	– the path flow
floatType cost	– the path cost.
floatType* buffer	– a working space for outside using.
short markStatus	– the mark status.
vector<TNM_SLINK∗ > path	– store the links traversed by the path.

Constructor:

void TNM_SPATH() – a default constructor

Return – none.

Operations:

bool IsConnected() – check if the links are connecting with each other.

Return – true if connecting exists; false otherwise

TNM_SNODE* GetStartNode() – get the start node of the path

Return – the pointer of the start node.

TNM_SNODE* GetEndNode() – get the end node of the path

Return – the pointer of the end node.

floatType PathLength() – compute path length

Return – the length of the path.

floatType PathCostS() – compute static path cost by simply adding the link-cost

Return – the static cost.

floatType PathCost(bool fbToll = false) – compute path cost by going through its all links

Return – the cost.

largeInt TDPathTime(int startTime, largeInt simHorz) – compute time dependent path travel time

Return – the travel time.

floatType TDPathCost(int startTime, largeInt simHorz) – compute time dependent path travel cost

Return – the cost.

floatType FDPathCost(bool fbToll = false) – the first order derivative of static path cost

Return – the first order derivative of the cost.

void ToLinkFlow() – convert path flow to links flows

Return – none.

void SetID() – set path ID

Return – none.

void SetID(int id) – enforce an ID to the path

Return – none.

void UnsetID() – set ID = 0

Return – none.

int GetLinkNum() – get the number of links traversed by the path

Return – the number of links.

bool FormCycle(TNM_SLINK *link) – check if the input link will form a cycle with the existing links on path

Return – true if a cycle forms; false otherwise.

void ToLinkOrgFlow(floatType flow) – update for each link volume as well origin-based flow

Return – none.

void AugmentFlow() – add flow to all links belong to the path with the minimum cost

Return – none.

void Print(bool node = false) – print out the information of the path

Return – none.

CLASS TNM_PROBPATH PARENT CLASS: TNM_SPATH

Data Members:

TNM_PPATH* m_ppath	– a TNM_PPATH object, through which to get the distribution of the path travel time.

Constructor:

void TNM_ProbPath() – a default constructor

Return – none.

Operations:

bool Setup(TNM_PPATH *pPath) – set up the data member pointer m_ppath

Return – true if succeeds; false otherwise.

TNM_PPATH* GetPPath() – get the distribution of the path travel time

Return – the pointer of the data member m_ppath

Overridables:

void Print(bool node = false) – print out the information of the path

Return	– none.

CLASS TNM_PPATH   PARENT CLASS: N/A

Data Members:

TNM_ProbDistPath m_dist	– distribution of path travel time.
floatType* stochTD	– store the third order stochastic data.
TNM_PRBLK* m_via	– which link the path at a specific node point to.
TNM_PPATH* m_nextPath	– which path the path at a specific node point to.
vector$<$TNM_PPATH$*$ $>$ m_upstreamPath	– the set of path that is created based on the current path.

Constructor:

void TNM_PPATH() – a default constructor.

Return	– none.

CLASS TNM_PROBDIS   PARENT CLASS: N/A

Data Members:

floatType* m_spt	– the array of support points.
floatType* m_pdf	– the array of probability corresponding to the support points.
int m_numEffSpt	– the number of effective support points.
static int m_sptRes	– the resolution of support points, default = 100.
static double m_prbRes	– probability points resolution, default = 0.01;.
static int m_count	– count how many instances.
static int m_sdAnchor	– number of points for which SD is checked.

Constructor:

void TNM_ProbDis() – a default constructor.

Return	– none.

Operations:

int InitializeFromCDF(floatType *cdf, int np, double minSupport, floatType maxSupport) – initialize the distribution of the given CDF

Return – the number of effective support points.

void Reset(floatType b, floatType e, floatType prob $= -1.0$) – reset the value of the minimum and maximum support points, and set that all probabilities equal 0.01 (there are 100 support points)

Return – none.

int Convolution(TNM_ProbDist *pb) – convolution of two distributions

Return – 0

int FSDCheck(TNM_ProbDist *pb) – check the first order stochastic dominance (FSD) between two random variables

Return – 1 if "this" dominates, -1 otherwise, 0 if no dominance established.

floatType CumProb(floatType sprt) – calculate the cumulative probability: get P such that P= P(x<sprt)

Return – P.

floatType InvCumProb(floatType prob) – get sprt such that P(x<sprt) = prob

Return – sprt.

void GetCDF(floatType *&cdf, floatType *&bp, int n) – return a CDF of n+1 points

Return – none.

inline floatType* GetPdfPtr() – get the array of PDF

Return – m_pdf.

inline floatType* GetSptPtr() – get the array of the support points

Return – m_spt.

inline floatType GetPdf(int i) – get a specific value of PDF

Return – m_pdf[i] if 0 ≤i<m_numEffSpt; -1 otherwise.

inline floatType GetSpt(int i) – get a specific support point

Return – m_spt[i] if 0 ≤i<m_numEffSpt; -POS_INF_FLOAT otherwise.

inline floatType GetMinSupport() – get the minimum support point

Return – m_spt[0].

inline floatType GetMaxSupport() – get the maximum support point

Return – m_spt[m_numEffSpt −1].

inline floatType GetMinBreakPoint() – get the minimum break point

Return – $1.5 \cdot$m_spt[0] $- 0.5 cdot$m_spt[1].

inline floatType GetMaxBreakPoint() – get the maximum break point

Return – $1.5 \cdot$m_spt[m_numEffSpt −1] − $0.5 \cdot$m_spt[m_numEffSpt−2].

inline int GetNumEffSupport() – get the number of effective support points

Return – m_numEffSpt.

void GetMeanVar(floatType &mean, floatType &variance) – get the mean and variance of the distribution

Return – none.

floatType CumProb(int ix, floatType cump, floatType sprt) – get the probability of being less a given value

Return – P(x<sprt).

floatType InvCumProb(int ix, floatType cump, floatType prob) – get sprt such that P(x<sprt) = prob

Return – sprt.

static int GetSptRes() – get the resolution of the support points

Return – m_sptRes.

static floatType GetProbRes() – get the resolution of probability

Return – m_prbRes.

static int GetSDAnchor() – get the number of points used for FSD check

Return – m_sdAnchor.

static void SetSDAnchor(int sa) – set the number of points used for FSD check

Return – none.

void Consolidate() – call this function to consolidate the distribution

Return – none.

Class TNM_ProbDistPath   Parent class: TNM_ProbDis

Data Members:

floatType* m_cdfArea	– a array of areas under the cumulative density function (CDF), used to check the second order stochastic dominance (SSD) only.

`Constructor:`

void TNM_ProbDistPath() – a default constructor

Return	– none.

`Operations:`

void ProduceCDFArea() – generate a array of m_cdfArea, specially used for SSD check

Return	– none.

floatType GetCDFArea(int i) – get the area under the CDF before a given point

Return	– the area under the CDF.

floatType* GetCDFArea() – get the data member m_cdfArea

Return	– m_cdfArea

void ClearCDFArea() – erase the memory occupied by m_cdfArea

Return	– none.