Some Implementation Issues in Approximate Algorithms for Reliable *A Priori* Shortest Path Problem

Xing Wu
Department of Civil and Environmental Engineering
Northwestern University
2145 Sheridan Road, Evanston IL 60208

Yu (Marco) Nie *
Department of Civil and Environmental Engineering
Northwestern University, 2145 Sheridan Road, Evanston, IL 60208
Email: y-nie@northwestern.edu
Phone: 1-847-467-0502

August 1, 2008

Submitted to *Transportation Network Modeling* Committee **(ADB30)**
for Presentation at the 88th Annual Meeting of Transportation Research Board and
for Publication in the Journal of Transportation Research Record

Word counts based on Latex: 5655 words + 4 tables + 2 figures = 7155

*Corresponding author

Abstract

This paper studies solution techniques for the problem of finding *a priori* paths that are shortest to ensure a specified probability of on-time arrival in a stochastic network. A new discretization scheme called α-discrete is proposed. The scheme better suites large-scale applications because it does not depend on problem-specific parameters. A procedure for evaluating convolution integrals based on the new scheme is given and its complexity is analyzed. Other implementation strategies are also discussed in order to improve the computational performance of the exact yet non-polynomial label-correcting algorithm. These include an approximate method based on extreme-dominance and two cycle avoidance strategies. Extensive numerical experiments are conducted to test the effects of the proposed implementation strategies using different networks and different types of distributions.
Keywords: A priori shortest path, discretization, label-correcting, extreme-dominance

1 Introduction

Optimal path problems in a stochastic network have been intensively studied. Conventionally, a path is considered optimal if it incurs the least expected travel time (LET) [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. In order to address the reliability of path travel times, Frank [13] (also [14]) defined the optimal path as the one that maximizes the probability of realizing a travel time equal to or less than a given threshold. [15] suggested using the maximum probability of being the shortest path as an optimality index. Employing expected utility theory, [16] showed that, for polynomial functions, utility maximization is reduced to a class of bi-criteria shortest path problems that trade off mean and variance of random travel times. This result is consistent with the mean-variance rule that has long been used in portfolio selection [17]. Similar instances of utility-based routing problems were studied in [18, 19, 20]. Stochastic optimal path problems have also been approached using robust optimization, which usually implies that a path is optimal if its worst-case travel time is the minimum [21, 22]. When correlations between travel time are considered, the robust routing problems are NP-hard even under restrictive assumptions [21]. [23] studied a robust optimization model for general network flow problems without considering correlations, and proposed a polynomial algorithm to find robust shortest paths. For stochastic and time-varying networks, [24] defined the optimal path as the one that realizes the least possible travel time. Later [25] and [26] explored other definitions of optimality based on various dominance relationships, namely, deterministic dominance, the first-order stochastic dominance (FSD) and expected value dominance. Label-correcting algorithms were proposed to solve non-dominant paths corresponding to each definition of dominance rules. [27] also used FSD to determine optimal paths in a stochastic network, and proposed a network reduction algorithm for acyclic networks.

Based on Frank's optimality [13], [28] studied the *reliable a priori shortest path problem* (RASP), which aims to find *a priori* paths that are shortest to ensure a specified probability of on-time arrival. [28] showed that the RASP problem belongs to a class of multi-criteria shortest path problems [29, 16, 7], which rely on a dominance relationship to obtain Pareto-optimal solutions. Since the dominance relationship in the RASP problem is defined with respect to the cumulative distribution function (CDF) of path travel times, it is effectively equivalent to the FSD rule considered in [25] and [27]. The RASP formulation proposed in [28] is continuous and, unlike in previous studies such as [25] and [5], defines the optimality with respect to arrival rather than departure time. The latter property makes it simpler to account for time-varying nature of travel time distributions. A label-correcting algorithm similar to that of [25] was proposed to solve the RASP problem.

This paper aims to propose and test several implementation strategies intended to improve the computational performance of the solution algorithms for the RASP problem. Since the dominance relationship is determined based on the CDF of path travel times, how to calculate and store these CDFs is critical to the efficiency of solution algorithms. These operations often involves discretizing continuous probability density functions and numerically evaluating convolution integrals. A challenge in the conventional discretization scheme (e.g. [30, 28]) is that the length of the analysis period T has to be set so large that trips on most paths can be completed with a probability of 1.0. For one thing, it is difficult to determine T with analytical methods. More importantly, T is problem-specific in the sense that it increases with network size and depends on travel time distributions. Because computational cost increases rapidly with T for the same resolution, the existing discretization scheme is not suitable for large networks. This paper proposes an alternative scheme to overcome the drawback, based on the inverse of CDF. A procedure to evaluate convolution integral using

this discretization scheme is also proposed.

It is well known that multi-criteria shortest path problems are intractable because the number of non-dominant paths does not have a polynomial bound. The RASP problem is no exception. Typical heuristic strategies intended to improve computational performance limit the size of non-dominant paths. However, these strategies may not yield correct Pareto-optimal solutions. In the worse case, they may not even identify a subset of non-dominant paths [25]. As a recent example, *extreme-dominance approximation* (EDA) strategy was proposed in [28]. EDA removes non-dominant paths which do not contribute directly to the Pareto frontier, thereby effectively restricting the number of these paths. Preliminary results in [28] demonstrated the satisfactory performance of EDA. However, a more extensive computational study is needed to evaluate EDA on networks of different sizes and densities, and with the new discretization scheme.

[28] proved the acyclicity of non-dominant paths and suggested that preventing cyclic paths from temporarily entering the non-dominance set may be beneficial from a computational point view. However, no numerical results were provided in [28]. In this study, two different cycle avoidance strategies are proposed and experiments were conducted to verify their impacts on the overall computational performance.

The remainder of the paper is organized as follows. Section 2 reviews the reliable *a priori* shortest path problems and a general label-correcting algorithm. Section 3.1 first reviews the discretization scheme adopted in [28] and then proposes a new one. Section 3.2 and 3.3 discuss the extreme-dominance approximation and cycle avoidance strategies, respectively. Section 4 presents a comprehensive computational study. The last section concludes the paper.

2 Problem statement and solution algorithm

Consider a directed and connected network $G(\mathcal{N}, \mathcal{A}, \mathcal{P})$ consisting of a set of nodes $\mathcal{N}$ ($|\mathcal{N}| = n$), a set of links $\mathcal{A}$ ($|\mathcal{A}| = m$), and a probability distribution $\mathcal{P}$ describing the statistics of link travel times. The analysis period is set to $[0, T]$. Let the destination of routing be s and the desirable arrival time be aligned with the end of the analysis period T. The travel 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 $p_{ij}(\cdot)$. Let $F_{ij}(\cdot)$ be the cumulative distribution function (CDF) of c_{ij}. For a more focused discussion on implementation issues, the dependence of p_{ij} on time of day and correlations among different c_{ij} are ignored in this paper [1]. 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 r and s form a set K^{rs}. Finally, let $u_k^s(b)$ denote the maximum probability of arriving at s through path k^{rs} no later than T, departing r with a time budget b.

This paper is concerned with *the reliable a priori shortest path problem* (RASP) which aims to find, starting from any node $i \neq s$, *a priori* paths which are shortest to ensure a specified probability of arriving the destination s on time. [28] showed that this problem is equivalent to finding all non-dominant paths under the first-order stochastic dominance rule. In what follows, we briefly summarize the results of [28]. The reader is referred to that work for more details.

Let us first define the first-order stochastic dominance (FSD) for the formulation of the RASP problem. To our best knowledge, this concept was first introduced to shortest path problems in [25, 26], albeit in a form different from our presentation.

[1] The dependence on time and correlations are addressed in [28] and [31]. We note that most solution techniques discussed in this paper are applicable in those extended models.

Definition 1 (First-order stochastic dominance (FSD) $\succ_1$) *Path k^{rs} dominates path l^{rs} in the first order, denoted as $k^{rs} \succ_1 l^{rs}$, if 1) $u_k^{rs}(b) \geq u_l^{rs}(b), \forall b \in [0,T]$; and 2) $\exists$ at least one open interval $\Lambda \subset [0,T]$ with nonzero Lebsegue measure such that $u_k^{rs}(b) > u_l^{rs}(b), \forall b \in \Lambda$.*

Non-dominant paths under the FSD rule are called *FSD-admissible* paths in this paper, as defined below.

Definition 2 (FSD-admissible path) *A path l^{rs} is FSD-admissible if $\exists$ no path k^{rs} such that $k^{rs} \succ_1 l^{rs}$.*

The RASP problem equals the problem of identifying all FSD-admissible paths between $(i,s), \forall i \neq s$. However, it is possible that an FSD-admissible path is not shortest for any on-time arrival probability. To clarify this point, we define FSD optimality in the following.

Definition 3 (FSD-optimal path) *A path k^{rs} is FSD-optimal if 1) it is FSD-admissible and 2) $\exists$ one open interval $\Lambda \subset [0,T]$ with nonzero Lebsegue measure such that $u_k^{rs}(b) \geq u_l^{rs}(b), \forall b \in \Lambda, \forall l \neq k$.*

We shall denote the set of FSD-admissible and FSD-optimal paths between the OD pair rs with Γ^{rs} and Ω^{rs}, respectively. Note that $\Omega^{rs} \subseteq \Gamma^{rs}$ by definition. At any node $i \in \mathcal{N}$, define $u^{is}(b) \equiv \max\{u_k^{is}(b), \forall k^{is} \in \Omega^{is}, \forall b\}$. The function $u^{is}(\cdot)$ is called *Pareto frontier function* at node i, which constitutes optimal solutions of the RASP problem. Note that, once u^{is} is known, one can identify a path $\bar{k}^{is}(b) \in \Omega^{is}$ such that $u_{\bar{k}}^{is}(b) = u^{is}(b)$ for a given b.

The above definitions employ the function $u_k^{rs}(\cdot)$ to represent the distribution of the random path travel time π_k^{rs}. We note that this distribution may also be represented by the inverse of u_k^{rs}, denoted by $v_k^{rs}(\cdot)$. $v_k^{rs}(\alpha)$ gives the shortest travel time b (or the latest departure time $T-b$) to arrive at s at or earlier than T with a probability α. According to Definition 1, if two paths such that $k^{rs} \succ_1 l^{rs}$, then $v_k^{rs}(\alpha) \leq v_l^{rs}(\alpha), \forall \alpha \in [0,1]$ and $v_k^{rs}(\alpha) < l_l^{rs}(\alpha)$ for some α. FSD optimality and Pareto frontier function can be redefined using v_k^{rs} accordingly. In particular, the *inverse Pareto frontier function* $v^{is}(\alpha) \equiv \min\{v_k^{is}(\alpha), \forall k^{is} \in \Omega^{is}, \forall \alpha\}$. One reason why using v_k^{rs} to represent the distribution of π_k^{rs} may be more favorable is that it is defined on a fixed support $[0,1]$, whereas the support of u_k^{rs} depends on T, which varies with problem-specific parameters such as network size and distributions. We shall elaborate on this point in the next section.

[25] showed that any subpath of an FSD-admissible path must also be FSD-admissible. Using this result, [28] formulated the RASP problem as the following dynamic programming problem:

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

where $k^{js} \diamond ij$ extends path k^{js} along link ij; $\gamma_{\succ}^1(K)$ represents the operation which retrieves FSD-admissible paths from a set K using Definition 2; 0^{ss} is a dummy path representing the boundary condition. Problem (1) can be solved using a label correcting algorithm. The following algorithm, named FSD-LC, is taken from [28] with slight modifications [2].

Algorithm FSD-LC

Step 0 Initialization. Let 0^{ss} be a dummy path from the destination to itself. Initialize the scan list $Q = \{0^{ss}\}$. set $\pi_0^{ss} = 1$ with probability 1.

Step 1 Select the first path from Q, denoted as l^{js}, and delete it from Q.

Step 2 For any predecessor node i of j, create a new path k^{is} by extending l^{js} along link ij.

[2] Conceptually, the algorithm is similar to Miller-Hooks's algorithm [25].

step 2.1 Calculate the distribution of π_k^{is} from the distribution of π_l^{js} by convolution integral (see Section 3.1).

step 2.2 Compare the new path k^{is} with current *Pareto frontier*. If the frontier is dominated by k^{is}, update the frontier with the distribution of π_k^{is}, drop all existing FSD-admissible paths at node i, and set $\Gamma^{is} = \{k^{is}\}, \Omega^{is} = \{k^{is}\}$. Otherwise, further compare the distribution of the new paths and all existing FSD-admissible paths to check FSD admissibility. If any of the existing path dominates k_{is}, drop k_{is} and go back to Step 2; otherwise, delete all paths that are dominated by k_{is} from Γ^{is}, set $\Gamma^{is} \cup \{k_{is}\}$, and update $Q = Q \cup \{k^{is}\}$.

Step 3 If Q is empty, stop; otherwise go to Step 1.

3 Implementation issues in Algorithm FSD-LC

Generally, Algorithm FSD-LC has a non-polynomial complexity because the number of FSD-admissible paths may grow exponentially with network size [25, 28]. However, the actual performance of the algorithm depends on a number of implementation issues, which are the focal points of the present paper. We first examine in the next section how discretization schemes impact the evaluation of the distribution of π_k^{rs} in step 2.1 of Algorithm FSD-LC and their cross-comparison in step 2.2.

3.1 Discretization schemes

If random link travel times follow a continuous probability density function p_{ij}, the distribution of path travel time π_k^{rs} can be calculated recursively from the following convolution integral

$$u_k^{is}(b) = \int_0^b u_k^{js}(b-w)p_{ij}(w)dw, \ \forall b \in [0,T] \tag{2}$$

The above integral may be calculated using Laplace Transform (LT) [32]. However, because efficient LT implementation requires evaluating convolution only at a few pre-determined Gaussian quadrature points, the method may only identify a small subset of all FSD-admissible paths and thereby fail to determine correct Pareto frontier functions. The LT-based method is also numerically unstable since it has to deal with the inverse of a Vandermonde matrix [32].

A simple yet effective alternative that overcomes the above difficulties is to discretize the analysis period $[0,T]$ evenly into L intervals of length ϕ, and check the distribution for FSD-admissibility at all L points. In such a discretization scheme, the probability density function p_{ij} has to be first discretized to get the corresponding probability mass function P_{ij}, namely

$$P_{ij}(b) = \begin{cases} \int_b^{b+\phi} p_{ij}(w)dw & b = 0, \phi, ..., (L-1)\phi \\ \int_b^{\infty} p_{ij}(w)dw & b = L\phi \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Accordingly, the evaluation of convolution integral in (2) is replaced with a finite sum as follows:

$$u_k^{is}(b) = \sum_0^b u_k^{js}(b-\phi)P_{ij}(\phi), \ \forall b = 0, \phi, \cdots, L\phi \tag{4}$$

Note that $O(L^2)$ steps are required to calculate $u_k^{is}(b)$ for all discrete b using (4). The above discretization scheme is adopted in [28], and hereafter referred to as b-discrete method. When using the b-discrete method, [28] showed that Algorithm FSD-LC runs in a non-polynomial time $O(mn^{2n-1}L + mn^nL^2)$. To see this, note that in the worst case, there are n^{n-1} paths and therefore steps 2.1 and 2.2 of the algorithm has to be executed for mn^{n-1} times. As analyzed before, step 2.1 requires $O(L^2)$ steps and it takes $O(n^{n-1}L)$ steps to compare the distribution function of the newly generated path with those of the paths currently stored in Γ^{is} (recall that the distribution function is approximated by L discrete points).

The size of L depends on both T and ϕ. Although ϕ can be set independent of network size, T is not. Note that in our model T is the desired arrival time. Ideally, the analysis period $[0, T]$ should equal the longest possible time to arrive at the destination starting at any time $t \geq 0$ at any origin. Essentially, this allows all trips to be completed with a probability up to 1.0. For example, if the desired arrival time is 9:00 am and the longest possible travel time is 6 hours, the origin of the analysis time period should be set at 3:00 am. In this case, if $\phi = 5$ minutes, $L = 6 \times 12 = 72$. 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 [24] 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 folowing. 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

$$b_t^{ij} = F_{ij}^{-1}(t\epsilon), \quad t = 1, \cdots, L \tag{5}$$

where $F_{ij}^{-1}(\cdot)$ is the inverse CDF of c_{ij}. Equation (5) implies

$$\int_{b_{t-1}^{ij}}^{b_t^{ij}} p_{ij}(w)dw = \epsilon, \quad t = 1, \cdots, L, \tag{6}$$

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

$$p_{ij}(\hat{b}_t^{ij})(b_t^{ij} - b_{t-1}^{ij}) = \epsilon \tag{7}$$

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

$$P_{ij}(\hat{b}_t^{ij}) = \epsilon, \quad t = 1, \cdots, L \tag{8}$$

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(4) 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.

3.2 Extreme-dominance approximation

Because the number of FSD-admissible path may grow exponentially, it is necessary in large-scale applications to restrict the size of admissible sets to make Algorithm FSD-LC computationally viable. In [25], the number of admissible paths is not allowed to exceed a pre-determined upper bound and any extra admissible paths will be removed, either arbitrarily or according to some heuristic rule. [28] proposed an approximate algorithm based on the assumption that dynamic programming applies to FSD-optimal paths. In this method, paths that are not FSD-optimal are excluded from further consideration. In other words, a path is retained only if it contributes to the Pareto frontier function. This heuristic method is refereed to as *extreme-dominance approximate* (FSD-EDA) following [33]. FSD-EDA offers a much better complexity compared to FSD-LC because it limits the number of admissible path to be L. In the following, we will show how this heuristic method is implemented using α-discrete method (b-discrete can be implemented similarly, cf. [28]). Let $\sigma(l_k^{is})$ denote the total number of discrete points $\alpha = t\epsilon$ at which $v_k^{is}(\alpha) = v^{is}(\alpha)$. Thus, if $\sigma(l_k^{is}) = 0$, path l_k^{is} is not FSD-optimal. Also, recall that $\bar{k}^{is}(\alpha)$ is the path associated with the inverse Paretio frontier at α.

Procedure FSD-CHECK

Inputs: a new path l^{is}, a set of current FSD-admissible paths Γ_{is}, and Pareto frontier function v^{is}.

Return: a Boolean value FSD indicating whether or not l^{is} is FSD-admissible

Step 0 Set FSD = TRUE, set $\sigma(l^{is}) = 0$, set $Q' = \emptyset$ (Q' is the set of paths that are currently FSD-admissible but not FSD-optimal).

Step 1 Update Pareto frontier and identify Q'.

for each $\alpha = 0, \epsilon, ..., L\epsilon$ do:

set $k^{is} = \bar{k}^{is}(\alpha)$. If $v_l^{is}(\alpha) < v^{is}(\alpha)$: update $v^{is}(\alpha) = v_l^{is}(\alpha), \bar{k}^{is}(\alpha) = l^{is}, \sigma(l^{is}) = \sigma(l^{is}) + 1, \sigma(k^{is}) = \sigma(k^{is}) - 1$, If $\sigma(k^{is}) = 0$, set $Q' = Q' \cup k^{is}$.

end for

Step 2 If **Extreme-Dominance**, go to Step 3; otherwise, go to Step 4.

Step 3 Delete all paths in Q'. If $\sigma(l^{is}) = 0$, return FALSE; otherwise; return TRUE.

Step 4 While LR = TRUE and Q' is not empty, do

for each path k^{is} in Q' do: set $n_l = 0, n_e = 0, n_g = 0$.

for $\alpha = 0, \epsilon, ..., L\epsilon$ and if ($n_l = 0$ or $n_g = 0$) do

if $v_l^{is}(\alpha) < v_k^{is}(\alpha)$, set $n_g = n_g + 1$; else if $v_l^{is}(\alpha) = v_k^{is}(\alpha)$, $n_e = n_e + 1$; else, $n_l = n_l + 1$.

end for

if $n_l = 0$, set LR = FALSE; else if $n_g = 0$, set $\Gamma_{is} = \Gamma_{is}/k^{is}$.

end while

As shown in the procedure, the most time-consuming pairwise comparison of path distribution functions (Step 4), which consumes at most $O(Ln^{n-1})$ steps, is avoided in extreme-dominance approximation. Since the total number of FSD-admissible paths cannot exceed L, the complexity of FSD-EDA is pseudo-polynomial, namely, $O(mL^3)$ for b-discrete and $O(mL^3 \log(L))$ for α-discrete. Although it is computationally appealing, FSD-EDA does not necessarily obtain correct Pareto frontier functions. In the worse case, it is unclear if the algorithm can ensure to identify at least a subset of FSD-admissible paths. Preliminary results reported in [28] indicate that FSD-EDA is able to produce good approximations of Pareto frontier functions despite its theoretical deficiency. In this paper, we will further test the validity of FSD-EDA using different networks and discretization schemes.

3.3 Cycle check

[28] proved that FSD-admissible paths must not contain any cycles. This property also holds in the time-dependent case provided that time-varying probability density functions satisfy certain stochastic first-in-first-out condition [31]. Acyclicity can be utilized in Algorithm FSD-LC to screen a new path generated from Step 2. Specifically, whenever path k^{js} is extended to node i, one should check if i is already contained in k^{js}. This extra operation may be worthwhile, because once a cyclic path is allowed to enter step 2.1 and 2.2, eliminating it may take up to $O(L^2 + n^{n-1}L)$ steps. Let $\omega(l^{js})$ be a subpath operator, namely, $\omega(l^{js}) = l^{ps}$ and $jp \in \mathcal{A}$. A complete cycle check can be performed as follows.

Procedure CYCLE-CHECK

Inputs: path l^{js} and node i such that $ij \in \mathcal{A}$.

Return: a boolean value **CR** indicating whether or not i has been traversed by l^{js}.

Step 0 Set $l^{ps} = \omega(l^{js})$, if $p = i$, **CR=TRUE**, stop; else if $p = s$, **CR=FALSE**, stop; otherwise go to step 1.

Step 1 Set $j = p$, go to Step 0.

The above operation will consume at most $O(n)$ steps. In sparse networks, a lot of cycles are direct, namely, they involve only adjacent links (e.g. $i \rightarrow j \rightarrow i$). Therefore, checking only for these direct cycles is still able to eliminate many if not most cyclic paths but is more computationally efficient. Implementation of this heuristic cycle check is same to the above procedure except that Step 1 is ignored. However, since it does not exclude all cyclic paths, whether this approximate method will improve overall computation performance remains to be verified using numerical results.

4 Numerical results

Comprehensive numerical experiments are conducted in this section to: 1) compare different discretization schemes (α-discrete vs b-discrete), 2) examine the sensitivity of the practical performance of Algorithms FSD-LC and FSD-EDA to network size and density, and 3) test the impact of various cycle check strategies. The algorithms were coded using MS-C++ and tested on a Windows-XP (64) workstation with two 3.00 GHz Xeon CPUs and 4G RAM.

4.1 Comparison of two discretization schemes

In this section, Algorithm FSD-LC is implemented using both α-discrete and b-discrete schemes and tested on a real road network of Chicago area, which has 933 nodes and 2930 links. The network, known as *Chicago Sketch*, has been used to test traffic assignment problems [34] [3].

Link distributions in this experiment are assumed to follow either Gamma or uniform distribution. While real travel times may follow neither distribution, the purpose is to reveal the impact of the shape of the distribution function on performance of the discrete methods. The probability density function of Gamma distribution is:

$$p_{ij}(x) = \frac{1}{\theta^{\kappa}\Gamma(\kappa)} x^{\kappa-1} e^{-x/\theta} \tag{9}$$

where θ and κ are parameters and $\Gamma(\cdot)$ is the Gamma function. The mean and variance of a Gamma distribution are $\kappa\theta$ and $\kappa\theta^2$, respectively. In our experiment, parameters κ and θ are generated randomly using a uniform distribution for all links. Specifically, $\theta \propto U(0.8, 3.5)$, and $\kappa \propto U(1.0, 2.5)$. Thus, the mean and standard deviation of link traversal times are within the ranges $[0.80, 8.75]$ and $[0.80, 5.53]$, respectively. For uniform distribution $p_{ij}(x) = 1/(U - L)$, L is fixed at 0 and U is randomly drawn from $[3.5, 10]$. The length of the analysis period T is set to 100 for b-discrete, which we found through trial-and-error was large enough to guarantee trips on admissible paths to be completed with a probability close enough to 1.0.

Four different b-discrete schemes are considered, in which T is divided into $L = 100,\ 200,\ 500$ and 1000 discrete intervals (correspondingly $\phi = 1,\ 0.5,\ 0.2$ and 0.1). These are named as Schemes B-I, B-II, B-III and B-IV respectively. In addition, we tested three α-discrete schemes: $\epsilon = 2\%,\ 1\%$ and 0.5% corresponding to $L = 50, 100$ and 200, named as Schemes A-I, A-II and A-III. Because B-IV has highest resolution in all schemes, it will be used as the benchmark scheme, against which approximation errors of other schemes will be evaluated.

We first ran Algorithm FSD-LC to find FSD-admissible paths for destination 933 using each of the seven schemes. Pareto frontier functions of path travel times between O-D pair 1-933 are compared in Figure 1. Note that the inverse Pareto functions generated from α-discrete have been inverted for comparison. As expected, the higher the resolution of a discretization scheme, the closer its Pareto frontier is to the benchmark. Interestingly, in all cases, the approximation errors tend to underestimate the on-time arrival probability. This is a good news for risk-averse travelers since it keeps errors on the safe side. Secondly, for the same L, approximation errors from α-discrete seem smaller. Figures 1(b) and 1(d) show that the frontier produced by A-III ($L = 200$) is very close to the benchmark for either distribution. However, α-discrete leads to relatively larger errors when desired probabilities are close to 1.0 or 0.0. In Figure 1(b), for example, the frontier of Scheme A-I is comparable to that of B-III when $0.1 < \alpha < 0.6$, whereas

[3] The reader is referred to http://www.bgu.ac.il/ bargera/tntp/ for a detailed description of the network data.

much larger discrepancy between the two is observed beyond that range. The reason may be that some $\hat{b}_t^{ij}$ (cf. Equation (5)) are overestimated (underestimated) when $t\epsilon$ is close to 0 (1). This phenomenon is more prominent in Gamma distribution, probably because of its long tail.

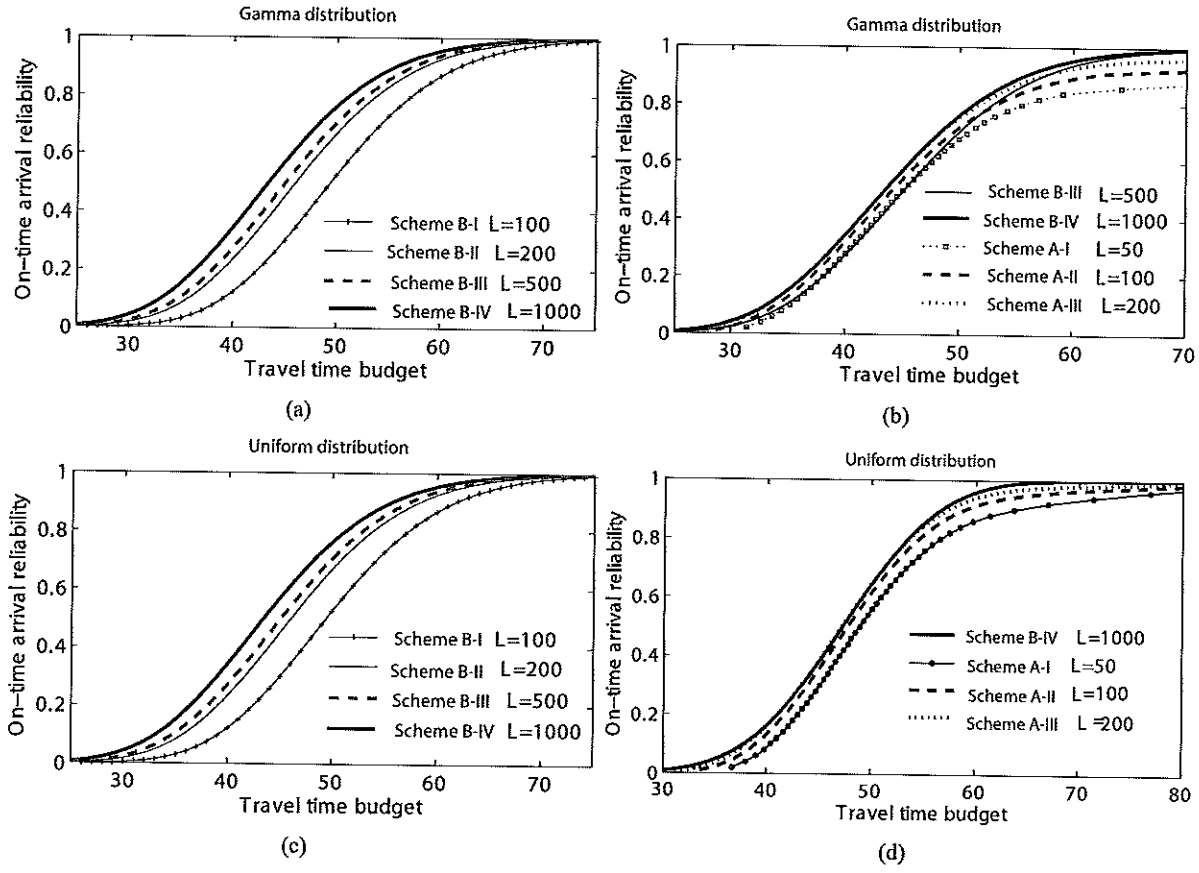


Figure 1: Comparison of Pareto frontiers between O-D pair $(1, 933)$ (using Scheme B-IV as benchmark)

For further comparison, Algorithm FSD-LC was run for 10 randomly-selected destinations, and the average performance indexes for each scheme are reported in Table 1. To quantify the discrepancy between the frontiers of Schemes x and y, we define *overall maximum gap* as

$$\Delta_{xy}^{s} = \sup\left\{ \max(|u_x^{is}(b) - u_y^{is}(b)|, \forall b \in \Lambda), \forall i \in \mathcal{N} \right\} \tag{10}$$

where $u_x^{is}(\cdot)$ is the Pareto frontier function between i and s for Scheme x. Since Figure 1 suggests that the relative discrepancy may vary for different on-time probabilities, we calculated the gaps separately in two different intervals of T, namely $\Lambda_1 = [0, 60]$ and $\Lambda_1 = (60, 100)$.

The results reported in Table 1 confirm our previous observation that with the same L, the α-discrete method produces more accurate frontiers on average. A possible explanation is that α-discrete has more "effective" support points on average to represent link distributions in our experiments. In b-discrete, a support point at b is not effective when $F(b)$ is close enough to 1.0 or 0.0. We found such ineffective support points can be as many as more than 80 out of 100 in b-discrete. In contrast, most discrete points in α-discrete are effective.

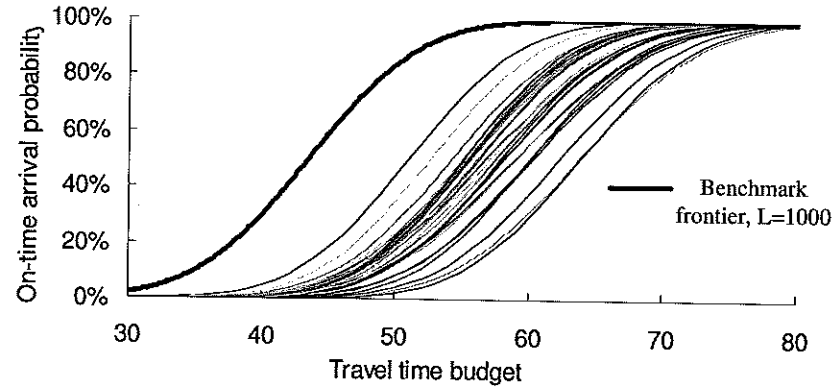
Table 1: Computational performance of seven scenarios, when the link traversal times follow a gamma distribution or a uniform distribution

Schemes	B-I	B-II	B-III	B-VI	A-I	A-II	A-III
		b-discrete				α-discrete	
Gamma distribution							
CPU time	1.45	6.94	43.54	165.51	6.65	33.92	180.46
Ave. $\lvert\Gamma^{is}\rvert$	5.25	6.67	7.40	7.14	3.32	3.62	4.08
Max. $\lvert\Gamma^{is}\rvert$	23.80	31.50	34.20	31.70	19.90	21.00	23.50
$\Lambda_1 : T \leq 60$							
Ave. Δ^s_{xy}	0.291	0.134	0.034		0.063	0.040	0.020
Max. Δ^s_{xy}	0.462	0.217	0.055		0.257	0.085	0.059
$\Lambda_2 : 60 < T < 100$							
Ave. Δ^s_{xy}	0.116	0.047	0.011		0.063	0.053	0.022
Max. Δ^s_{xy}	0.464	0.211	0.052		0.254	0.123	0.066
Uniform distribution							
cpu time	4.54	19.16	157.96	506.57	1.98	10.29	41.29
Ave. $\lvert\Gamma^{is}\rvert$	10.67	13.48	16.29	16.08	1.67	1.68	1.71
Max. $\lvert\Gamma^{is}\rvert$	44.10	58.40	58.40	60.90	8.40	7.90	7.70
$\Lambda_1 : T \leq 60$							
Ave. $\Delta^s_{x,IV}$	0.336	0.159	0.041		0.045	0.035	0.013
Max. $\Delta^s_{x,IV}$	0.528	0.255	0.066		0.192	0.062	0.040
$\Lambda_2 : 60 < T < 100$							
Ave. $\Delta^s_{x,IV}$	0.142	0.058	0.013		0.045	0.030	0.011
Max. $\Delta^s_{x,IV}$	0.557	0.266	0.068		0.190	0.093	0.044

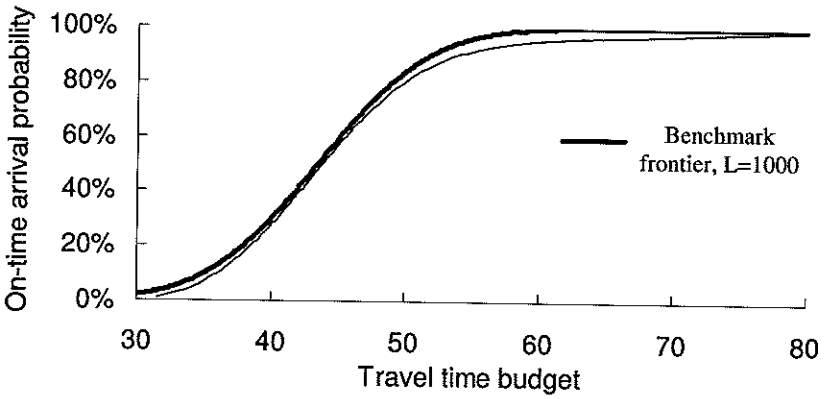
Note: 1) all frontiers are compared with the frontier of Scenario IV; 2) CPU times are measured in second; 3) Ave. $\lvert\Gamma^{is}\rvert$ (or Max. $\lvert\Gamma^{is}\rvert$) refers to the average (or maximum) number of FSD-admissible paths for all nodes.

For b-discrete, when L is increased from 100 to 200, 500 and 1000, the CPU times increase by 4.8, 30.0 and 114.1 times (Gamma distribution) or 4.2, 34.8 and 111.6 times (uniform distribution) respectively. For α-discrete, the CPU times increase by 5.1 and 27.1 times (Gamma distribution) or 5.2 and 20.9 times (uniform distribution) respectively when L is increased from 50 to 100 and then to 200. These trends roughly agree with the previous analysis which indicates the complexity increases with $O(L^2)$ (for b-discrete) and $O(L^2 \log L)$ (for α-discrete). Note that because the number of FSD-admissible paths also grows slightly when L increases, the actual grow rate of CPU times is higher than the prediction of the above bounds.

For Gamma distribution, b-discrete runs faster than α-discrete for similar L values. In part, this is because Procedure ACI needs extra $O(\log(L))$ steps for sorting. Another reason has to do with an implementation detail. In the b-discrete method, the comparison between CDFs (conducted in Procedure FSD-CHECK) is terminated if 1) any of them gets close enough to 1, and 2) a dominance relationship holds. Since these conditions could be met well before b reaches the upper bound T, the cost of path comparisons may be lower in b-discrete than α-discrete if L is same.



(a) b-discrete method



(b) α-discrete method

Figure 2: CDFs of FSD-admissible paths between (547, 722) (L = 100)

Interestingly, the bottom panel of Table 1 shows seemingly conflicting results, namely *α-discrete* method ran consistently faster. A close look reveals that for uniform distribution, the number of FSD-admissible paths solved by α-discrete is much less than that by b-discrete. In order to reveal why this so happens, we plotted in Figure 2(a) the distribution functions of 42 FSD-admissible paths between (547, 722) generated

from Scheme B-I, together with the frontier of the benchmark scheme B-IV. In comparison, only one FSD-admissible path is identified from Scheme A-II (see Figure 2(b)). Figure 2(a) shows that within the interval $0.01 \leq \alpha \leq 0.99$, there is only one non-dominant path which is the one found also by α-discrete. Other 41 FSD-admissible paths are non-dominant either in the interval $0.99 < \alpha < 1$ (40 paths) or the interval $0 < \alpha < 0.01$ (1 path). Because no support point exists in the two intervals in A-II (note that $\epsilon = 0.01$), those 41 non-dominant solutions are missed in α-discrete. We note that, however, these missed paths are almost useless in practice as few travelers would be sensitive to an improvement of less than 1% on-time arrival reliability. More importantly, the frontier generated from Scheme A-II, with only one admissible path, turn out to be much more accurate than that from Scheme B-I which consist of 42 admissible paths.

To summarize, our experiments indicate that the α-discrete method is a competitive alternative to the traditional b-discrete method. It not only makes it possible to apply Algorithm FSD-LC to large networks, but also provides reasonable approximation with comparable computational expense.

4.2 Impact of network size and density

This experiment is designed to test how network size (n, number of nodes) and density (m/n, where m is the number of links) affect the performance of FSD-LC and FSD-EDA. A random network generator is used to create a sequence of grid networks of various sizes (see Column 1 of Table 2 for n and m of those networks). Gamma distribution is used to model random travel times, with the same setting of θ and κ from Section 4.1. We only use α-discrete ($\epsilon = 1\%$) in this experiment because b-discrete is computationally demanding in larger networks. For example, it takes more than 2 hours to finish a single run using b-discrete for a 50 by 50 network, in which both T and L have to be set to 500 in order to achieve reasonable approximation.

For each network shown in Table 2, Algorithms FSD-LC and FSD-EDA were run 10 times for randomly selected destinations. Average performance indexes are reported in Table 2. The overall maximum gap Δ^s_{xy} defined in Equation (10) is used to measure the difference between the Pareto frontiers produced by Algorithms FSD-EDA and FSD-LC (note that x and y in Equation (10) now refer to the different algorithms instead of different schemes).

First of all, the experiment suggests that for either algorithm $|\Gamma_{is}|$ does not increase exponentially with n. This observation agrees with results reported in previous studies (e.g. [25]). Actually, $|\Gamma_{is}|$ almost grows linearly with n in our test. The experiment also shows that the relationship between CPU times and n can be fitted using a quadratic function for both algorithms. Specifically,

$$\text{CPU}_{\text{LC}} = 0.0003n^2 - 0.5623n + 184.97(sec), \quad \text{CPU}_{\text{EDA}} = 0.00005n^2 - 0.0506n - 14.944(sec),$$

Secondly, Table 2 indicates that Algorithm FSD-EDA outperformed the exact algorithm in CPU times, especially for large networks. Note that FSD-EDA is almost 5 times faster on average for the 70×70 network. Apparently, this computational advantage is achieved because FSD-EDA operates on a smaller set of FSD-admissible paths. Nevertheless, despite it ignores a large number of admissible paths (for the 70×70 network it identifies only 1/4 of all admissible paths), FSD-EDA generates good approximates of the Pareto-frontier. Note that in all test scenarios, the maximum gap between the frontiers is always smaller than 0.001, corresponding to no more than 0.1% of on-time arrival probability.

We next examine the impact of network density (m/n) on the performance of Algorithm FSD-LC. Networks with different m are generated by adding extra links into the 20×20 and 30×30 gird networks from the first experiment (see Columns 1 and 6 in Table 3). Similarly, 10 destinations are randomly selected for each network and solved using Algorithm FSD-LC to obtain average performance indexes. In addition

Table 2: Computational performance of FSD-LC and FSD-EDA for networks of various sizs

Network	Algorithm	CPU time	Ave. $\lvert\Gamma_{is}\rvert$	Max. $\lvert\Gamma_{is}\rvert$	Ave. Δ^s_{xy}	Max. Δ^s_{xy}
70 × 70	FSD-LC	5665.10	92.37	2013.20	4.23E-04	6.47E-04
#links: 19320	FSD-EDA.	1087.17	23.46	163.90		
60 × 60	FSD-LC	2301.55	56.45	1051.50	3.28E-04	5.34E-04
#links: 14160	FSD-EDA.	651.92	18.29	117.30		
50 × 50	FSD-LC	921.93	39.30	751.70	4.97E-04	6.77E-04
#links: 9800	FSD-EDA.	343.75	15.67	102.90		
40 × 40	FSD-LC	261.61	18.12	253.70	2.07E-04	3.82E-04
#links: 6240	FSD-EDA.	136.31	9.59	65.20		
30 × 30	FSD-LC	82.62	10.52	118.56	2.22E-04	4.32E-04
#links: 3480	FSD-EDA.	53.24	6.67	40.89		
20 × 20	FSD-LC	14.04	4.38	27.11	1.18E-04	1.87E-04
#links: 1520	FSD-EDA.	11.72	3.65	17.78		
10 × 10	FSD-LC	1.46	2.09	6.89	2.92E-05	3.66E-05
#links: 360	FSD-EDA.	1.42	2.00	6.22		

Note: Δ^s_{xy} refers to the overall maximum gaps between two frontiers solved by FSD-LC and FSD-EDA.

Table 3: Computational performance of FSD-LC algorithm for networks with various densities

20 × 20 network					30 × 30 network				
#links	CPU time	#path compa.	Ave. $\lvert\Gamma^{is}\rvert$	Max. $\lvert\Gamma^{is}\rvert$	#links	CPU time	#paths compa.	Ave. $\lvert\Gamma^{is}\rvert$	Max. $\lvert\Gamma^{is}\rvert$
1250	12.39	17109	3.86	25.20	3480	48.90	114017	6.63	60.00
3120	68.51	85185	5.83	33.70	6180	240.86	474984	9.67	89.11
4668	143.02	177388	5.50	35.00	10648	369.60	490837	6.65	60.00
5972	125.36	123372	4.98	31.50	13772	429.78	498049	6.61	49.00
6996	112.83	99452	4.34	24.50	16476	342.13	350482	5.88	38.67

to CPU times and sizes of admissible paths, the numbers of cross-comparison between paths (Step 4 in Procedure FSD-LC) are also reported in Table 3 (Columns 3 and 8). The most interesting finding from Table 3 is that CPU times and the number of FSD-admissible paths ($|\Gamma^{is}|$) do not increase monotonically with m. Instead, these indexes first increased but then began to drop once m exceeded a certain threshold. For the 20 × 20 network, when m increases from 4668 to 5972, both $|\Gamma^{is}|$ and CPU times are reduced by nearly 10%. $|\Gamma^{is}|$ initially increases when the network becomes denser, apparently because more paths are available in a denser network. However, the increase in density also has a counter effect: more shortcuts will appear in a denser network which may dominate many otherwise admissible paths. As a result, while the total number of paths always increases, $|\Gamma^{is}|$ may not.

4.3 Effects of cycle check

As mentioned in Section 3.3), cycle check introduces extra costs but could potentially reduce the number of path comparisons. This experiment attempts to identify the effects of various cycle check strategies on the

Table 4: Comparison of various cycle-check strategies

	Full cycle check			Direct cycle check			No cycle check	
	CPU time	#path compa.	#cycles avoided	cpu time	#path compa.	#cycles avoided	cpu time	#path compa.
#links	**Panel 1: five 20×20 networks**							
1250	14.33	19497	2307	14.18	19507	2302	20.17	25168
3120	51.53	57648	2897	51.29	57797	2823	58.51	65026
4668	83.29	86041	3329	83.67	86956	2872	91.03	94528
5972	79.63	73832	2023	79.36	74163	1857	84.04	78342
6996	70.20	56207	1786	69.42	56341	1719	73.86	59959
#links	**Panel 2: five 30×30 networks**							
3480	49.44	113980	7384	48.90	114017	7365	68.20	133173
6180	241.10	473219	16146	240.86	474984	15268	280.73	547067
10648	368.76	487593	13407	369.60	490837	11790	399.77	536163
13772	430.36	495286	12530	429.78	498049	11152	458.19	535445
16476	344.82	350078	8666	342.13	350482	8464	363.43	372348
#nodes	**Panel 3: six grid networks with different number of links (as in Table 2))**							
10×10	1.47	1124	220	1.46	1124	220	2.03	1566
20×20	13.57	20598	2175	14.04	20607	2170	19.08	25548
30×30	79.61	329720	11944	78.86	329927	11840	110.19	364970
40×40	264.14	2455149	40772	261.61	2455400	40658	369.85	2670260
50×50	952.04	18572455	123443	922.33	18572757	123296	1289.61	19349234
60×60	2357.32	71435620	529929	2301.55	69375765	536156	3296.20	75604585

performance of Algorithm FSD-LC. Two strategies are considered: *full cycle check* in which all nodes on the current path are checked to make sure the new path does not form a cycle, and *direct cycle check* which attempts to only exclude the direct cycle involving adjacent links (see Section 3.3). These two strategies are compared with the no-cycle-check scenario on three sets of networks: two sets of random networks with various densities (20×20 and 30×30, panels 1 and 2 in Table 4), and a set of grid networks with similar density but different sizes (panel 3 in Table 4). Algorithm FSD-LC is used in this test with a discretization scheme A-II. Table 4 compares the average performance indexes obtained from 10 runs for each network.

Overall, the expense of cycle check was well paid off. Table 4 shows that the algorithm was consistently accelerated by either cycle check strategy, with up to 30% CPU time savings. Moreover, the two strategies demonstrated very similar performance, apparently because more than 90% of excluded cycles are direct cycles regardless of network size and density. In larger networks ($n > 2000$), direct cycle check appears to lead full cycle check with a modest margin (2% - 3%). Thus, it may be sufficient to perform a direct cycle check in practice.

The third panel in Table 4 reveals that the relative CPU time saving of cycle check is quite stable when network size increases but the density does not change ($m/n \simeq 4$). In all 6 networks (with number of nodes increasing from 100 to 3600), Algorithm FSD-LC ran about 30% faster with cycle check. In the first and second panels, however, the relative performance gain from cycle check seems to drop as the networks become denser. For example, in the 30×30 networks, the percentage of CPU time saving decreases from about 30% (when $m = 3480$) to only 5% (when $m = 16476$). Thus, the relative CPU time saving obtained

by cycle check seems to depend on network density rather than network size.

5 Conclusions

This paper examines a few implementation issues of approximate algorithms for the reliable a priori shortest path (RASP) problem. We proposed a new discretization scheme called α-discrete, as well as a corresponding procedure to evaluate convolution integrals. This new scheme avoids the dependence on problem-specific parameters such as network size and topology, and thereby better suites large-scale applications. We also discussed implementation strategies intended to improve the computational performance of the existing label-correcting algorithms. These include an approximate method based on extreme dominance and two cycle check strategies. Extensive numerical experiments were conducted to test the effects of the aforementioned implementation strategies. Findings from these experiments are summarized as follows:

1. Generally, α-discrete produces better approximation results with comparable computational costs. It typically identifies a smaller set of FSD-admissible paths. However, most of missed paths appear not to substantially impact resulting Pareto frontier functions and therefore be of little impact in practical applications.
2. Extreme dominance approximation offers significant computational benefits. The errors from this approximation seem to be negligible for most practical purposes.
3. The number of FSD-admissible paths increases almost linearly with network size. The algorithm runs in a polynomial time. In particular, we found that the relationship between the consumed CPU time and the number of nodes can be fitted with a quadratic function.
4. Cycle check helps improve the overall computational performance. Moreover, it suffices to exclude direct cycles that involve only consecutive links, because they constitute most of the existing cycles regardless of network size and density. The test also shows that the relative benefit of cycle check depends on network density rather than network size.

[28] showed that the b-discrete method allows one to decompose the time dimension and thus reduce a time-dependent problem into a static one. However, this property no longer holds for α-discrete because the support points of the distributions do not correspond to a departure time interval like in b-discrete. As a result, developing efficient implementations of α-discrete in the time-dependent environment is a challenging issue that warrants further investigation. Another direction for further research is to compare the extreme-dominance approximation with other heuristic methods, such as those proposed in [25].

References

[1] R. W. Hall, "The fastest path through a network with random time-dependent travel time," *Transportation Science*, vol. 20, pp. 182–188, 1986.

[2] G. Andreatta and L. Romeo, "Stochastic shortest paths with recourse," *Networks*, vol. 18, pp. 193–204, 1988.

[3] G. H. Polychronopoulos and J. N. Tsitsiklis, "Stochastic shortest path problems with recourse," *Networks*, vol. 27, no. 2, pp. 133–143, 1996.

[4] R. K. Cheung, "Iterative methods for dynamic stochastic shortest path problems," *Naval Research Logistics*, vol. 45, pp. 769–789, 1998.

[5] L. Fu and L. R. Rilett, "Expected shortest paths in dynamic and stochastic traffic networks," *Transportation Research B*, vol. 32, pp. 499–516, 1998.

[6] L. Fu, "An adaptive routing algorithm for in-vehicle route guidance systems with real-time information," *Transportation Research B*, vol. 35, no. 8, pp. 749–765, 2001.

[7] E. D. Miller-Hooks and H. S. Mahmassani, "Least expected time paths in stochastic, time-varying transportation networks," *Transportation Science*, vol. 34, no. 2, pp. 198–215, 2000.

[8] E. D. Miller-Hooks, "Adaptive least-expected time paths in stochastic, time-varying transportation and data networks," *Networks*, vol. 37, no. 1, pp. 35–52, 2001.

[9] S. T. Waller and A. K. Ziliaskopoulos, "On the online shortest path problem with limited arc cost dependencies," *Networks*, vol. 40, pp. 216–227, 2002.

[10] J. S. Provan, "A polynomial-time algorithm to find shortest paths with recourse," *Networks*, vol. 41, no. 2, pp. 115–125, 2003.

[11] Y. Fan, R. Kalaba, and J. Moore, "Shortest paths in stochastic networks with correlated link costs," *Computers and Mathematics with applications*, vol. 49, pp. 1549–1564, 2005.

[12] S. Gao and I. Chabini, "Optimal routing policy problems in stochastic time-dependent networks," *Transportation Research B*, vol. 40, no. 2, pp. 93–122, 2006.

[13] H. Frank, "Shortest paths in probabilistic graphs," *Operations Research*, vol. 17, no. 4, pp. 583–599, 1969.

[14] P. B. Mirchandani, "Shortest distance and reliability of probabilistic networks," *Computers and Operations Research*, vol. 3, pp. 347–355, 1976.

[15] C. E. Sigal, A. Alan, B. Pritsker, and J. J. Solberg, "The stochastic shortest route problem," *Operations Research*, vol. 28, no. 5, pp. 1122–1129, 1980.

[16] R. P. Loui, "Optimal paths in graphs with stochastic or multidimensional weights," *Communications of the ACM*, vol. 26, pp. 670–676, 1983.

[17] H. Markowitz, "Portfolio selection," *Journal of Finance*, vol. 7, pp. 77–91, 1952.

[18] A. Eiger, P. B. Mirchandani, and H. Soroush, "Path preferences and optimal paths in probabilistic networks," *Transportation Science*, vol. 19, no. 1, pp. 75–84, 1985.

[19] I. Murthy and S. Sarkar, "A relaxation-based pruning technique for a class of stochastic shortest path problems," *Transportation Science*, vol. 30, pp. 220–236, 1996.

[20] I. Murthy and S. Sarkar, "Stochastic shortest path problems with piecewise linear concave linear functions," *Management Science*, vol. 44, pp. 125–136, 1998.

[21] G. Yu and J. Yang, "On the robust shortest path problem," *Computers and Operations Research*, vol. 25, pp. 457–468, 1998.

[22] R. Montemani and L. Gambardella, "An exact algorithm for the robust shortest path problem with interval data," *Computers and Operations Research*, vol. 31, pp. 1667–1680, 2004.

[23] D. Bertsimas and M. Sim, "Robust discrete optimization and network flow," *Mathematical Programming Series B*, vol. 98, pp. 49–71, 2003.

[24] E. D. Miller-Hooks and H. S. Mahmassani, "Least possible time paths in stochastic, time-varying networks," *Computers and Operations Research*, vol. 25, no. 2, pp. 1107–1125, 1998.

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

[26] E. D. Miller-Hooks and H. S. Mahmassani, "Path comparisons for a priori and time-adaptive decisions in stochastic, time-varying networks," *European Journal of Operational Research*, vol. 146, no. 2, pp. 67–82, 2003.

[27] J. Bard and J. Bennett, "Arc reduction and path preference in stochastic acyclic networks," *Management Science*, vol. 37(2), pp. 198–215, 1991.

[28] Y. Nie and X. Wu, "The shortest path problem considering on-time arrival probability," *Transportation Research Part B*, vol. Under Revision, 2008.

[29] P. Hansen, "Bicriterion path problems," in *Multiple criteria decision making theory and application* (G. Fandel and T. Gal, eds.), pp. 109–127, 1979.

[30] Y. Fan and Y. Nie, "Optimal routing for maximizing the travel time reliability," *Networks and Spatial Economics*, vol. 3, no. 6, pp. 333–344, 2006.

[31] Y. Nie and X. Wu, "Reliable a priori shortest path problem with limited spatial and temporal dependencies," *The 18th International Symposium on Transportation and Traffic Theory, under review*, 2008.

[32] Y. Fan, R. Kalaba, and J. Moore, "Arriving on time," *Journal of Optimization Theory and Applications*, vol. 127, pp. 497–513, 2005.

[33] M. I. Henig, "The shortest path problem with two objective functions," *European Journal of Operational Research*, vol. 25, pp. 281–291, 1985.

[34] H. Bar-Gera, "Origin-based algorithm for the traffic assignment problem," *Transportation Science*, vol. 36, pp. 398–417, 2002.