Implementation Issues for the Reliable A Priori Shortest Path Problem

Xing Wu and Yu (Marco) Nie

Solution techniques are studied 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 is well suited to 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 also are discussed to improve the computational performance of the exact yet nondeterministic polynomial label-correcting algorithm. These include an approximate method based on extreme dominance and two cycle-avoidance strategies. Comprehensive numerical experiments are conducted to test the effects of the proposed implementation strategies using different networks and different distribution types.

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 (*1–12*). To address the reliability of path travel times, Frank (*13*) and Mirchandani (*14*) define the optimal path as the one that maximizes the probability of realizing a travel time equal to or less than a given threshold. Sigal et al. suggest using the maximum probability of being the shortest path as an optimality index (*15*). Using expected utility theory, Loui shows 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 (*16*). This result is consistent with the mean-variance rule that has long been used in portfolio selection (*17*).

Similar routing problems have been studied elsewhere (*18–20*). Stochastic optimal-path problems also have been approached using robust optimization, which usually implies that a path is optimal if its worst-case travel time is the minimum (*21–23*). Miller-Hooks and Mahmassani define the optimal path as the one that realizes the least possible travel time in stochastic and time-varying networks (*24*). Later, other definitions of optimality based on various dominance relationships also were explored, namely, deterministic dominance, first-order stochastic dominance (FSD), and expected value dominance (*25, 26*). Label-correcting algorithms were proposed to solve nondominant paths corresponding to each definition of dominance rules. Bard and Bennett also used FSD to determine optimal paths in a stochastic network and proposed a network reduction algorithm for acyclic networks (*27*).

Department of Civil and Environmental Engineering, Northwestern University, 2145 Sheridan Road, Evanston, IL 60208. Corresponding author: Y. Nie, y-nie@northwestern.edu.

Transportation Research Record: Journal of the Transportation Research Board, No. 2091, Transportation Research Board of the National Academies, Washington, D.C., 2009, pp. 51–60.
DOI: 10.3141/2091-06

The reliable a priori shortest path problem (RASP) studied in this research aims to find a priori paths that are shortest to ensure a specified probability of on-time arrival. The authors (*28*) have shown that the RASP belongs to a class of multiple-criteria shortest path problems that rely on a dominance relationship to obtain Pareto-optimal solutions (*7, 16, 29*); that is, no further travel time improvements associated with any on-time arrival probability can be made without worsening those associated with other probability levels. Because the dominance relationship in RASP is defined with respect to the cumulative distribution function (CDF) of path travel times, it is effectively equivalent to the FSD rule considered by Miller-Hooks (*25*) and Bard and Bennett (*27*). The RASP formulation proposed by Nie and Wu (*28*) is continuous and solved with a label-correcting algorithm similar to that of Miller-Hooks (*25*).

This paper proposes and tests several implementation strategies intended to improve the computational performance of the solution algorithms for RASP. Because the dominance relationship is determined on the basis of CDFs, how to calculate and store them is critical to the efficiency of solution algorithms. These operations often involve discretizing continuous probability density functions and numerically evaluating convolution integrals. A challenge in the conventional discretization scheme (e.g., *28, 30*) 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 important, 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. An alternative scheme is proposed to overcome this drawback, using the inverse of a CDF. A procedure to evaluate the convolution integral using this discretization scheme is also proposed.

It is well known that multicriteria shortest path problems are intractable because of the nondeterministic polynomial (NP) bound of Pareto-optimal solutions. The RASP is no exception. Typical heuristic strategies attempt to overcome the difficulty by limiting the size of nondominant paths. For example, Nie and Wu recently proposed the extreme-dominance approximation (EDA) strategy (*28*). EDA ignores nondominant paths that do not contribute directly to the Pareto frontier, thereby effectively restricting the number of these paths. Preliminary results demonstrated the satisfactory performance of EDA (*28*). However, like other heuristics, this strategy may not yield correct Pareto-optimal solutions. In the worst case, it may not even identify a subset of nondominant paths. Therefore, a comprehensive computational study is needed to evaluate EDA on networks of different sizes and densities and with the newly proposed discretization scheme.

Nie and Wu proved the acyclicity of nondominant paths and suggest that preventing cyclic paths from temporarily entering the non-

dominance set may be beneficial from a computational point of view; however, no numerical results were provided (*28*). In this study, two cycle-avoidance strategies are proposed, and experiments are conducted to evaluate their impacts on the overall computational performance.

The remainder of the paper is organized as follows. First, the RASPs and a general label-correcting algorithm are reviewed. Next, the authors' previous discretization scheme is reviewed (*28*), and a better alternative is proposed. Then, EDA and cycle-avoidance strategies are discussed. Finally, a comprehensive computational study and conclusions are presented.

PROBLEM STATEMENT AND SOLUTION ALGORITHM

Consider a directed and connected network $G(\mathbf{N}, \mathbf{A}, \mathbf{P})$ consisting of a set of nodes $\mathbf{N}$ ($|\mathbf{N}| = n$), a set of links $\mathbf{A}$ ($|\mathbf{A}| = m$), and a probability distribution $\mathbf{P}$ describing the statistics of link travel times. The analysis period is set to be $[0, T]$. Let the destination of routing be s and the desirable arrival time be aligned with the end of the analysis period T. 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 CDF of c_{ij}. To focus discussion on implementation issues, the dependence of p_{ij} on time of day and correlations among c_{ij} are ignored in this paper; dependence on time and correlations are addressed elsewhere (*28*, *31*). Most solution techniques discussed in this paper are applicable in those extended models. 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 $u_k^{rs}(b)$ 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 RASP that aims to find, starting from any node $i \neq s$, a priori paths that are shortest to ensure a specified probability of arriving at the destination s on time. The authors showed that this problem is equivalent to finding all nondominant paths under the FSD rule, which is used to compare random variables based on their cumulative density functions (CDFs) (*28*). The results are summarized here, but readers are referred to the original work for more details.

First, FSD must be defined to formulate the RASP. To the authors' best knowledge, this concept was first introduced to shortest path problems by Miller-Hooks and Mahmassani, albeit in a form different from this one (*25*, *26*). Also, Definition 1 differs from the classic definition (*32*) because the random variables discussed herein (i.e., travel time or cost) are related to disutility instead of utility.

> Definition 1. FSD $\succ_1$: Path k^{rs} dominates path l^{rs} in the first order, denoted as $k^{rs} \succ_1 l^{rs}$, if and only if $u_k^{rs}(b) \geq u_l^{rs}(b)$ for all b in $[0, T]$ and at least one strict inequality. Nondominant 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 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, FSD optimality is defined next.
>
> Definition 3. FSD-optimal path: A path k^{rs} is FSD-optimal if (a) it is FSD-admissible and (b) $\exists$ one open interval $\Lambda \subset [0, T]$ with nonzero Lebesque measures such that $u_k^{rs}(b) \geq u_l^{rs}(b)$, $\forall b \in \Lambda$, $\forall l \neq k$.

The set of FSD-admissible and FSD-optimal paths between an OD pair rs can be with Γ^{rs} and Ω^{rs}, respectively. Note that $\Omega^{rs} \subseteq \Gamma^{rs}$ by definition. At any node $i \in \mathbf{N}$, 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 the Pareto frontier function at node i, which constitutes optimal solutions of the RASP. Note that after 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 use the function $u_k^{rs}(\cdot)$ to represent the distribution of the random path travel time π_k^{rs}. This distribution also may be represented by the inverse of u_k^{rs}, denoted by $v_k^{rs}(\cdot)$. The $v_k^{rs}(\alpha)$ term 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 are such that $k^{rs} \succ_1 l^{rs}$, then $v_k^{rs}(\alpha) \leq v_l^{rs}(\alpha)$ for all α in [0, 1] and $v_k^{rs}(\alpha) < v_l^{rs}(\alpha)$ for some α. FSD optimality and the Pareto frontier function can be redefined accordingly using v_k^{rs}. 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 range [0, 1], whereas the support of u_k^{rs} depends on T, which varies with problem-specific parameters such as network size and distributions. This point is elaborated in the next section.

Miller-Hooks shows that any subpath of an FSD-admissible path must also be FSD-admissible (*25*). Using this result, Nie and Wu formulated the RASP as the following dynamic programming problem (*28*):

Find $\Gamma^{is}, \forall i$ such that

$$\Gamma^{is} = \gamma_{\succ}^{1}(k^{is} = k^{js} \Diamond ij \mid k^{js} \in \Gamma^{js} \qquad \forall ij \in \mathbf{A}), \forall i \neq s; \Gamma^{ss} = 0^{ss} \qquad (1)$$

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

FSD-LC algorithm

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.

Calculate the distribution of π_k^{js} from the distribution of π_l^{js} by convolution integral.

Compare the new path k^{is} with the 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}, then 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.

IMPLEMENTATION ISSUES: FSD-LC ALGORITHM

The FSD-LC algorithm has an NP complexity because the number of FSD-admissible paths may grow exponentially with network size (*25, 28*). However, actual performance of the algorithm depends on many implementation issues, which are the focal points of the present paper. In the next section, the impact of discretization schemes on the evaluation of the π_k^{rs} distribution is examined in Step 2 of the FSD-LC algorithm.

Discretization Schemes

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

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

The above integral may be calculated using Laplace transform (LT) (*33*). However, because efficient LT implementation requires evaluating convolution only at a few predetermined Gaussian quadrature points, the method may only identify a small subset of all FSD-admissible paths and thereby fail to determine the correct Pareto frontier functions. The LT-based method is also numerically unstable because it must address the inverse of a Vandermonde matrix.

A simple yet effective alternative that overcomes these 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, p_{ij} must be first discretized to get the corresponding probability mass function (PMF) P_{ij}:

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

Accordingly, the evaluation of the convolution integral in Equation 2 is replaced with a finite sum as

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

Using Equation 4, $O(L^2)$ steps are required to calculate $u_k^{is}(b)$ for all discrete *b*. Nie and Wu adopted the above discretization scheme, hereafter referred to as the *b*-discrete method (*28*). When using *b*-discrete, the FSD-LC algorithm runs in an NP time $O(mn^{2n-1} + mn^n L^2)$. To see this, note that in the worst case, there are n^{n-1} paths and therefore Step 2 of the algorithm must be executed mn^{n-1} times. As analyzed before, convolution 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} (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* cannot. In the proposed 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, it allows all trips to be completed with a probability up to 1.0. For example, if the desired arrival time is 9:00 a.m. and the longest possible travel time is 6 h, then the origin of the analysis time period should be set at 3:00 a.m. In this case, if $\varphi = 5$ min, 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 (Miller-Hooks and Mahmassani discuss the least-possible time path problem [*24*]). To bypass this difficulty, one may simply set *T* to be a large number. However, this brute-force treatment will raise computational issues because the complexity of the discrete algorithm depends on *L*. In a nutshell, the *b*-discrete method is unsatisfactory because it leads to problem-specific complexity.

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

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

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

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

and with the mean value theorem, $\hat{b}_t^{ij}$ always can be found 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 PMF in this discrete scheme is given by

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

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

ACI procedure

Step 0. Set $\eta = 0$. For $t_1 = 1, \ldots, L$; for $t_2 = 1, \ldots, L$: 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 $v_k^{is}(t\epsilon) = z_{tL}$ $t = 1, \ldots, L$.

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 the procedure is dominated by the second step, which is higher than that of the discrete convolution (Equation 4) by a factor of $\log(L)$.

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

Extreme-Dominance Approximation

Because the number of FSD-admissible paths may grow exponentially, it is necessary in large-scale applications to restrict the size of admissible sets to make the FSD-LC algorithm computationally viable. Miller-Hooks does not allow the number of admissible paths to exceed a predetermined upper bound, and any extra admissible paths are removed, arbitrarily or according to a heuristic rule (*25*).

Nie and Wu's approximate algorithm is based on the assumption that dynamic programming applies to FSD-optimal paths (*28*). 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. Henig calls this heuristic method FSD–extreme-dominance approximate (FSD-EDA) (*34*). FSD-EDA offers much better complexity than FSD-LC because it limits the number of admissible paths to L.

The following display shows how this heuristic method is implemented using the α-discrete method (*b*-discrete can be implemented similarly [*28*]). Let $\sigma(k^{is})$ denote the total number of discrete points $\alpha = t\epsilon$, where $v_k^{is}(\alpha) = v^{is}(\alpha)$ [i.e., $v^{is}(\alpha)$ contributes to the frontier at α]. Thus, if $\sigma(k^{is}) = 0$, path k^{is} is not FSD-optimal. Also, recall that $\overline{k}^{is}(\alpha)$ is the path associated with the inverse Pareto frontier at α.

FSD-CHECK procedure
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 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 $\alpha = 0, \epsilon, 2\epsilon, \ldots, L\epsilon$ do
Set $k^{is} = \overline{k}^{is}(\alpha)$. If $v_l^{is}(\alpha) < v^{is}(\alpha)$: update
$v^{is}(\alpha) = v_l^{is}(\alpha)$, $\overline{k}^{is}(\alpha) = l^{is}$, $\sigma(l^{is}) = \sigma(l^{is}) + 1$, $\sigma(k^{is}) = \sigma(k^{is}) - 1$. 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 $\alpha = 0, \epsilon, 2\epsilon, \ldots, L\epsilon$ 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 $\Gamma^{is} = \dfrac{\Gamma^{is}}{k^{is}}$.
End while.

The most time-consuming pairwise comparison of distribution functions—Step 4, which consumes at most $O(Ln^{n-1})$ steps—is avoided in the procedure. Because the total number of FSD-admissible paths cannot exceed L, the complexity of FSD-EDA is pseudo-polynomial, that is, $O(mL^3)$ for *b*-discrete and $O(mL^3 \log L)$ for α-discrete. Although the procedure is computationally appealing, FSD-EDA does not necessarily obtain the correct Pareto frontier functions. In the worst case, it is unclear whether the algorithm can identify at least a subset of FSD-admissible paths. The authors' preliminary results indicate that FSD-EDA produces good approximations of Pareto frontier functions despite its theoretical deficiency (*28*). In this paper, the validity of FSD-EDA is further tested using different networks and discretization schemes.

Cycle Check

As proven previously, FSD-admissible paths must not contain any cycles (*28*). This property also holds in the time-dependent case, provided that time-varying probability density functions satisfy certain stochastic first-in, first-out conditions (*31*). Acyclicity can be used in the FSD-LC algorithm to inspect a new path generated from Step 2. Specifically, whenever path k^{js} is extended to node i, one should check whether i is already contained in k^{js}. This extra operation may be worthwhile, because once a cyclic path is allowed to enter Step 2 of the FSD-LC algorithm, 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 $jp \in \mathbf{A}$. A complete cycle check can be performed as follows.

CYCLE-CHECK Procedure
Inputs: Path l^{js} and node i such that $ij \in \mathbf{A}$.
Return: A Boolean value **CR** indicating whether 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.

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

NUMERICAL RESULTS

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

Comparison of Two Discretization Schemes

The FSD-LC algorithm was implemented using both α-discrete and *b*-discrete schemes and tested on a real road network in the Chicago area that has 933 nodes and 2,930 links. The network, known as Chicago Sketch, has been used to test traffic assignment problems (*35*).

Link distributions in this experiment are assumed to follow a Gamma or a uniform distribution. Although real travel times may follow neither distribution, the purpose is to reveal the impact of the shape of the distribution function on the performance of the discrete methods. The probability density function of the 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 Γ(·) is the Gamma function. The mean and variance of a Gamma distribution are κθ and $\kappa\theta^2$, respectively. In the present experiment, 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 for the *b*-discrete method is set to 100, which was found through trial and error to be large enough to guarantee trips on admissible paths to be completed with a probability close enough to 1.0.

Four *b*-discrete schemes are considered, in which T is divided into L = 100, 200, 500, and 1,000 discrete intervals (corresponding to φ = 1, 0.5, 0.2, and 0.1, respectively) named Schemes B-I, B-II, B-III, and B-IV, respectively. In addition, three α-discrete schemes were tested: ϵ = 2%, 1%, and 0.5%, corresponding to L = 50, 100, and 200 and named Schemes A-I, A-II, and A-III, respectively. Because B-IV has the highest resolution in all schemes, it was used as the benchmark scheme against which approximation errors of other schemes were evaluated.

The FSD-LC algorithm was run first to find FSD-admissible paths for Destination 933 using all seven schemes. Pareto frontier functions of path travel times for origin–destination (O-D) pair (1, 933) are shown, with inverse Pareto functions generated from the α-discrete method inverted for comparison, in Figure 1. As expected, the higher the resolution of a discretization scheme, the closer its Pareto frontier is to the benchmark. Interestingly, approximation errors tend to underestimate the on-time arrival probability in all cases. This finding is good news for risk-averse travelers because it keeps errors on the safe side. Second, for the same L, approximation errors from the α-discrete scheme seem smaller. The frontier produced by A-III (L = 200) is close to the benchmark for either distribution (Figures 1*b* and 1*d*). However, the α-discrete method leads to larger errors when desired probabilities are close to 1.0 or 0.0. For example, the frontier of Scheme A-I is comparable to that of B-III when $0.1 < \alpha < 0.6$, whereas a much larger discrepancy is observed between the two beyond that range. The reason may be that some $\hat{b}_t^{ij}$ (Equation 5) are overestimated (or underestimated) when $t\epsilon$ is close to 0 (or 1). This phenomenon is more prominent in Gamma distribution, probably because of its long tail.

For further comparison, the FSD-LC algorithm was run for 10 randomly selected destinations; average performance indexes for each scheme are reported in Table 1. To quantify the discrepancy between the frontiers of schemes x and y, overall maximum gap is defined as

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

where $u_x^{is}(\cdot)$ is the Pareto frontier function between i and s for scheme x. Because Figure 1 suggests that the relative discrepancy may vary for different on-time probabilities, the gaps were calculated separately in two intervals of T: $\Lambda_1 = [0, 60]$ and $\Lambda_2 = (60, 100)$.

Results reported in Table 1 confirm the authors' previous observations that, on average and with the same L, the α-discrete method produces more accurate frontiers. One possible explanation is that α-discrete has more "effective" support points to represent link distributions in the present experiments. In the *b*-discrete method, a

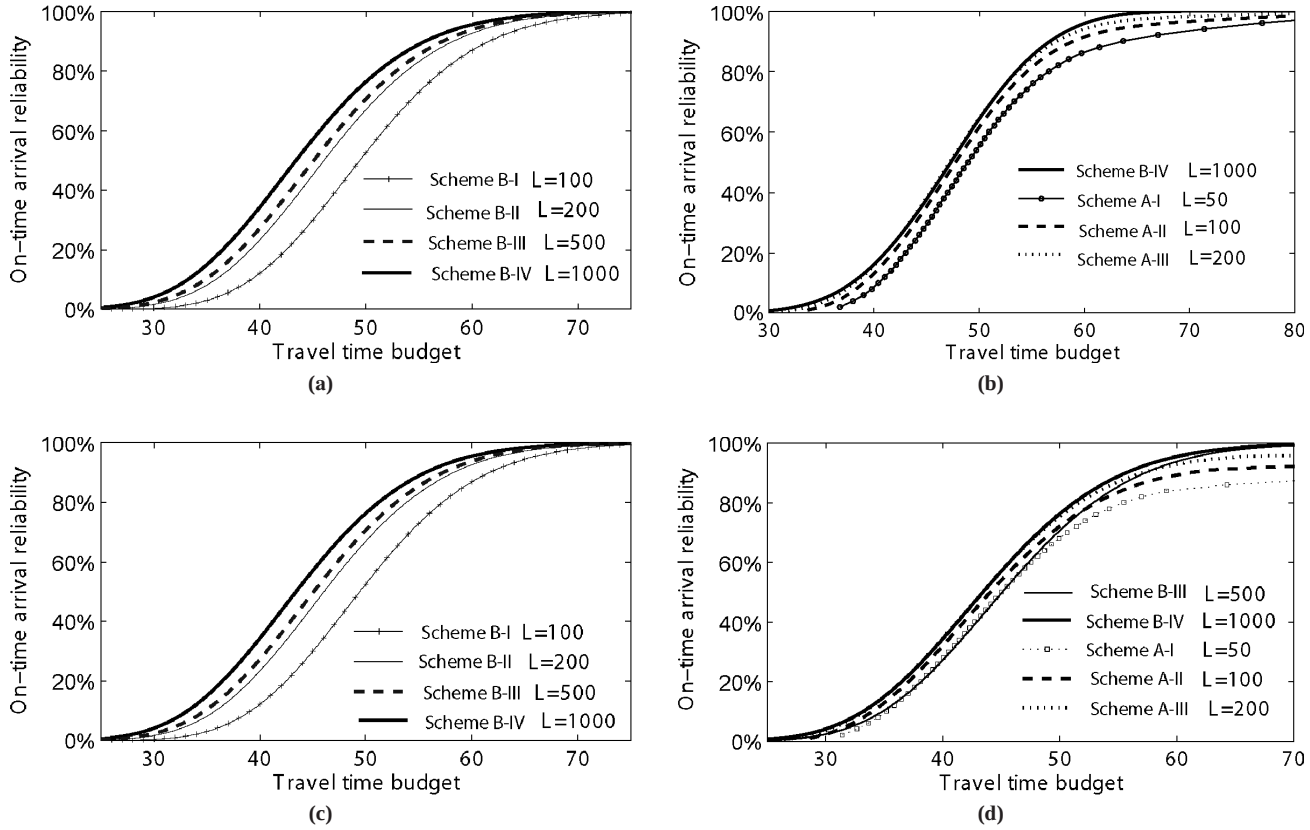

FIGURE 1 Comparison of Pareto frontiers for O-D Pair (1, 933) using Scheme B-IV as benchmark: (*a*, *b*) uniform distribution and (*c*, *d*) Gamma distribution.

TABLE 1 Computational Performance of Different Discretization Schemes on Chicago Sketch

	b-Discrete				α-Discrete		
Scheme	B-I	B-II	B-III	B-IV[a]	A-I	A-II	A-III
Gamma Distribution							
CPU time [b]	1.45	6.94	43.54	165.51	6.65	33.92	180.46
Avg $\lvert\Gamma^{is}\rvert$ [c]	5.25	6.67	7.40	7.14	3.32	3.62	4.08
Max $\lvert\Gamma^{is}\rvert$ [c]	23.80	31.50	34.20	31.70	19.90	21.00	23.50
	Λ_1: $T \le 60$						
Avg 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
	Λ_2: $60 < T < 100$						
Avg 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
Avg $\lvert\Gamma^{is}\rvert$	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
	Λ_1: $T \le 60$						
Avg $\Delta^s_{x,IV}$	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
	Λ_2: $60 < T < 100$						
Avg $\Delta^s_{x,IV}$	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

[a]All frontiers are compared with the frontier of Scenario B-IV.
[b]CPU times are measured in seconds.
[c]Avg $\lvert\Gamma^{is}\rvert$ (or max $\lvert\Gamma^{is}\rvert$) refers to the average (or maximum) number of FSD-admissible paths for all nodes.

support point at *b* is not effective when $F(b)$ is close enough to 1.0 or 0.0; in fact, more than 80 of 100 such support points can be ineffective in the *b*-discrete method. In contrast, most discrete points in α-discrete are effective.

For the *b*-discrete method, when *L* is increased from 100 to 200, 500, and 1,000, CPU times increase 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, when *L* is increased from 50 to 100 and then 200, CPU times increase 5.1 and 27.1 times (Gamma distribution) or 5.2 and 20.9 times (uniform distribution), respectively. These trends roughly agree with the previous analysis, which indicated that the complexity increases with $O(L^2)$ for *b*-discrete and $O(L^2 \log L)$ for the α-discrete method. 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 the Gamma distribution, the *b*-discrete method runs faster than α-discrete for similar *L* values, partly because the American Concrete Institute (ACI) procedure 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 the FSD-CHECK procedure) is terminated if any one gets close enough to 1 and a dominance relationship holds. Because these conditions could be met well before *b* reaches the upper bound *T*, the cost of path comparisons may be lower in the *b*-discrete method than in the α-discrete method if *L* is the same.

Interestingly, the uniform distribution results listed in Table 1 appear to be conflicting; specifically, the α-discrete method ran consistently faster. A close look reveals that for a uniform distribution, fewer FSD-admissible paths were solved by the α-discrete than by the *b*-discrete method. To reveal why this happens, the distribution functions of 42 FSD-admissible paths for O-D pair (547, 722) were plotted, generated from Scheme B-I, with the frontier of the benchmark scheme B-IV (Figure 2*a*). In comparison, only one FSD-admissible path is identified from Scheme A-II (Figure 2*b*). Figure 2*a* shows only one nondominant path within the interval $0.01 \le \alpha \le 0.09$—the path also found by the α-discrete method. Other 41 FSD-admissible paths are nondominant 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 Scheme A-II (note that $\epsilon = 0$), the α-discrete method misses those 41 nondominant solutions. However, those missed paths are almost useless in practice because few travelers would be sensitive to an improvement of less than 1% on-time arrival reliability. More important, with only 1 admissible path, the frontier generated from Scheme A-II turns out to be much more accurate than that from Scheme B-I, which consists of 42 admissible paths.

In summary, results indicate that the α-discrete method is a competitive alternative to the *b*-discrete method. It not only makes it possible to apply the FSD-LC algorithm to large networks but also provides a reasonable approximation with comparable computational expense.

Impact of Network Size and Density

This experiment was designed to test how network size (*n*, number of nodes) and density (*m*/*n*, where *m* is the number of links) affect FSD-LC and FSD-EDA performance. A random network generator was used to create a sequence of grid networks of various sizes (*n* and *m*

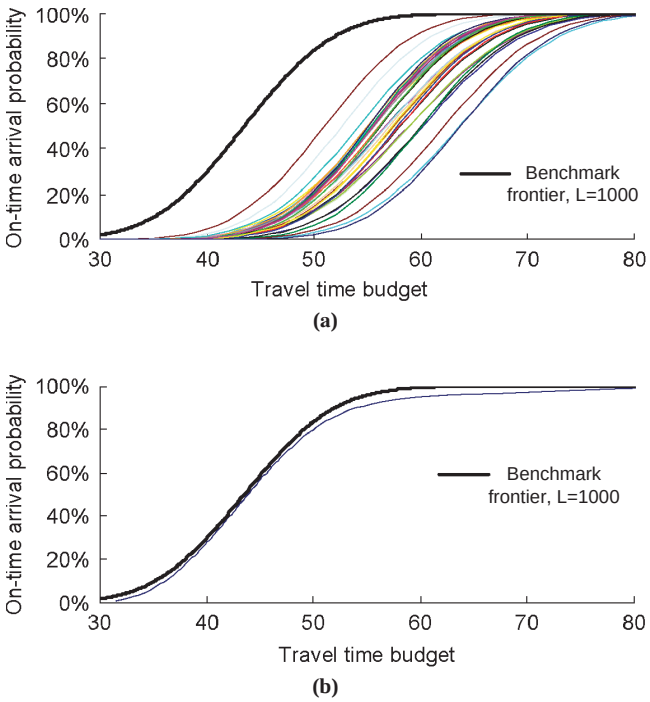

FIGURE 2 CDFs of FSD-admissible paths between O-D Pair (547, 722) (L = 100).

of those networks are listed in Table 2). Gamma distribution is used to model random travel times, with the same setting of θ and κ from the previous section. Only the α-discrete method ($\epsilon = 1\%$) was used in this experiment, because the *b*-discrete method is computationally demanding for large networks. For example, a single run requires more than 2 h to finish using *b*-discrete for a 50×50 network in which both T and L must be set to 500 to achieve a reasonable approximation.

For each network listed in Table 2, the FSD-LC and FSD-EDA algorithms 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 the two algorithms (note that x and y in Equation 10 now refer to different algorithms instead of different schemes).

First, experimental results suggest that for either algorithm, $|\Gamma^{is}|$ does not increase exponentially with n, in agreement with results reported in previous studies (*25*). Actually, $|\Gamma^{is}|$ grows almost linearly with n in the present test. Results also show that the relationship between CPU times and n can be fitted by using a quadratic function for both algorithms (measured in seconds). Specifically,

$$\text{CPU}_{\text{LC}} = 0.0003n^2 - 0.5623n + 184.97$$

$$\text{CPU}_{\text{EDA}} = 0.00005n^2 - 0.0506n - 14.944$$

The above formulae may be valid only for networks with similar topology.

Second, the FSD-EDA algorithm outperforms the exact algorithm in CPU times, especially for large networks (Table 2). It is almost five times faster, on average, for the 70×70 network. Apparently, this computational advantage is achieved because the FSD-EDA algorithm operates on a smaller set of FSD-admissible paths. Nevertheless, even though it ignores many admissible paths (e.g., it identifies only one-quarter of all admissible paths for the 70×70 network), the FSD-EDA algorithm generates good approximates of the Pareto frontier. In all test scenarios, the maximum gap between the frontiers is always less than 0.001, corresponding to no more than 0.1% of on-time arrival probability.

Next, the impact of network density (m/n) is examined on the performance of the FSD-LC algorithm. Networks with different m were generated by adding extra links to the 20×20 and 30×30 grid networks from the first experiment (Table 3). Similarly, 10 destinations were randomly selected for each network and solved using the FSD-LC algorithm to obtain average performance indexes. The CPU times, admissible path sizes, and number of cross-comparisons between paths also are reported in Table 3.

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 increase but then begin to drop once m exceeds a certain threshold. For the 20×20 network, when m increases from 4,668 to 5,972, both $|\Gamma^{is}|$ and CPU times are reduced by nearly 10%. $|\Gamma^{is}|$ increases initially when the

TABLE 2 Computational Performance of FSD-LC and FSD-EDA for Networks of Various Sizes

Network	Algorithm	CPU times	Avg $\lvert\Gamma^{is}\rvert$	Max $\lvert\Gamma^{is}\rvert$	Avg $\Delta^{s\,b}_{xy}$	Max Δ^s_{xy}
70×70[a]	FSD-LC	5,665.10	92.37	2,013.20	4.23E-04	6.47E-04
# links: 19,320	FSD-EDA	1,087.17	23.46	163.90		
60×60	FSD-LC	2,301.55	56.45	1,051.50	3.28E-04	5.34E-04
# links: 14,160	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: 9,800	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: 6,240	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: 3,480	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: 1,520	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		

[a] 70×70 grid network means there are 4,900 nodes in the network.
[b] 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	# Paths Compared	Avg $\|\Gamma^{is}\|$	Max $\|\Gamma^{is}\|$	# Links	CPU Time	# Paths Compared	Avg $\|\Gamma^{is}\|$	Max $\|\Gamma^{is}\|$
1,250	12.39	17,109	3.86	25.20	3,480	48.90	114,017	6.63	60.00
3,120	68.51	85,185	5.83	33.70	6,180	240.86	474,984	9.67	89.11
4,668	143.02	177,388	5.50	35.00	10,648	369.60	490,837	6.65	60.00
5,972	125.36	123,372	4.98	31.50	13,772	429.78	498,049	6.61	49.00
6,996	112.83	99,452	4.34	24.50	16,476	342.13	350,482	5.88	38.67

network becomes denser, apparently because more paths are available in a denser network. However, the increase in density also has a countereffect: more shortcuts will appear in a denser network, which may dominate many otherwise admissible paths. As a result, even though the total number of paths always increases, $|\Gamma^{is}|$ may not.

Effects of Cycle Check

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 performance of the FSD-LC algorithm. Two strategies are considered: full cycle check (all nodes on the current path are checked to make sure the new path does not form any cycle) and direct cycle check (which attempts only to exclude the direct cycle involving adjacent links). These two strategies were 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) and a set of grid networks with similar density but different sizes. The FSD-LC algorithm was used in this test with Discretization Scheme A-II. The average performance indexes obtained from 10 runs for each network are listed in Tables 4 and 5.

Overall, the expense of cycle check pays off well. The algorithm is consistently accelerated by either cycle-check strategy, with up to 30% CPU time savings (Tables 4 and 5). Moreover, the two strategies demonstrate similar performance, apparently because more than 90% of excluded cycles are direct cycles, regardless of network size and density. In larger networks ($n > 2{,}000$ nodes), direct cycle check appears to lead full cycle check with a modest margin (2% to 3%). Thus in practice, performing a direct cycle check may be sufficient.

The relative CPU time savings of cycle check is quite stable when the network size increases but the density does not change ($m/n \approx 4$; Tables 4 and 5). In all six networks (with the number of nodes increasing from 100 to 3,600), the FSD-LC algorithm runs about 30% faster with cycle check. However, this relative performance gain seems to drop as network density increases. For example, in the 30 × 30 networks, the percentage of CPU time savings decreases from about 30% (when $m = 3{,}480$) to only 5% (when $m = 16{,}476$). Thus, relative CPU time savings obtained by cycle check seems to depend on network density rather than network size.

CONCLUSIONS

A few implementation issues of approximate algorithms were examined for the RASP problem. A new discretization scheme called α-discrete was proposed, as well as a corresponding procedure for evaluating convolution integrals. This new scheme avoids a dependence on problem-specific parameters such as network size and topology and thereby better suits large-scale applications. Implemen-

TABLE 4 Comparison of Various Cycle-Check Strategies by Number of Links

	Full Cycle Check			Direct Cycle Check			No Cycle Check	
# Links	CPU Time	# Paths Compared	# Cycles Avoided	CPU Time	# Paths Compared	# Cycles Avoided	CPU Time	# Paths Compared
Panel 1. Five 20 × 20 Networks								
1,250	14.33	19,497	2,307	14.18	19,507	2,302	20.17	25,168
3,120	51.53	57,648	2,897	51.29	57,797	2,823	58.51	65,026
4,668	83.29	86,041	3,329	83.67	86,956	2,872	91.03	94,528
5,972	79.63	73,832	2,023	79.36	74,163	1,857	84.04	78,342
6,996	70.20	56,207	1,786	69.42	56,341	1,719	73.86	59,959
Panel 2. Five 30 × 30 Networks								
3,480	49.44	113,980	7,384	48.90	114,017	7,365	68.20	133,173
6,180	241.10	473,219	16,146	240.86	474,984	15,268	280.73	547,067
10,648	368.76	487,593	13,407	369.60	490,837	11,790	399.77	536,163
13,772	430.36	495,286	12,530	429.78	498,049	11,152	458.19	535,445
16,476	344.82	350,078	8,666	342.13	350,482	8,464	363.43	372,348

TABLE 5 Comparison of Various Cycle Check Strategies by Number of Nodes

	Full Cycle Check			Direct Cycle Check			No Cycle Check	
# Nodes	CPU Time	# Paths Compared	# Cycles Avoided	CPU Time	# Paths Compared	# Cycles Avoided	CPU Time	# Paths Compared
10×10	1.47	1,124	220	1.46	1,124	220	2.03	1,566
20×20	13.57	20,598	2,175	14.04	20,607	2,170	19.08	25,548
30×30	79.61	329,720	11,944	78.86	329,927	11,840	110.19	364,970
40×40	264.14	2,455,149	40,772	261.61	2,455,400	40,658	369.85	2,670,260
50×50	952.04	18,572,455	123,443	922.33	18,572,757	123,296	1,289.61	19,349,234
60×60	2,357.32	71,435,620	5,929	2,301.55	69,375,765	536,156	3,296.20	75,604,585

NOTE: Panel 3. Six grid networks with different number of links (as in Table 2).

tation strategies intended to improve the computational performance of the existing label-correcting algorithms (an approximate method based on extreme dominance and two cycle-check strategies) also were presented. Extensive numerical experiments were conducted to test the effects of these implementation strategies.

The findings from these experiments are summarized as follows:

- In general, the α-discrete method produces better approximation results than the *b*-discrete method with comparable computational costs. It typically identifies a smaller set of FSD-admissible paths, but most missed paths appear not to substantially affect the resulting Pareto frontier functions and therefore appear would be of little impact in practical applications.
- EDA offers significant computational benefits. The errors from this approximation seem to be negligible for most practical purposes.
- The number of FSD-admissible paths increases almost linearly with network size. The algorithm runs in polynomial time. In particular, the relationship between the consumed CPU time and the number of nodes can be fitted with a quadratic function for grid networks.
- Cycle check improves overall computational performance. In grid networks, it might be sufficient to exclude direct cycles because they seem to constitute most existing cycles. Also, the relative benefit of cycle check depends on the density rather than the size of the network.

The *b*-discrete method allows one to decompose the time dimension and thus reduce a time-dependent problem into a static one (*28*). However, this property does not hold for the α-discrete method because the support points of the distributions do not correspond to a departure time interval like in *b*-discrete. As a result, developing efficient α-discrete implementations in a time-dependent environment is a challenging issue that warrants further investigation. Another direction for future research is to compare EDA with other heuristic methods, such as those proposed by Miller-Hooks (*25*).

ACKNOWLEDGMENTS

This research was funded in part by the Federal Highway Administration's University Transportation Center (UTC) program through the Center for the Commercialization of Innovative Transportation Technology at Northwestern University.

REFERENCES

1. Hall, R. W. The Fastest Path Through a Network with Random Time-Dependent Travel Time. *Transportation Science*, Vol. 20, 1986, pp. 182–188.
2. Andreatta, G., and L. Romeo. Stochastic Shortest Paths with Recourse. *Networks*, Vol. 18, 1988, pp. 193–204.
3. Polychronopoulos, G. H., and J. N. Tsitsiklis. Stochastic Shortest Path Problems with Recourse. *Networks*, Vol. 27, 1996, pp. 133–143.
4. Cheung, R. K. Iterative Methods for Dynamic Stochastic Shortest Path Problems. *Naval Research Logistics*, Vol. 45, 1998, pp. 769–789.
5. Fu, L., and L. R. Rilett. Expected Shortest Paths in Dynamic and Stochastic Traffic Networks. *Transportation Research, Part B*, Vol. 32, No. 7, 1998, pp. 499–516.
6. Fu, L. An Adaptive Routing Algorithm for In-Vehicle Route Guidance Systems with Real-Time Information. *Transportation Research, Part B*, Vol. 35, No. 8, 2001, pp. 749–765.
7. Miller-Hooks, E. D., and H. S. Mahmassani. Least Expected Time Paths in Stochastic, Time-Varying Transportation Networks. *Transportation Science*, Vol. 34, 2000, pp. 198–215.
8. Miller-Hooks, E. D. Adaptive Least-Expected Time Paths in Stochastic, Time-Varying Transportation and Data Networks. *Networks*, Vol. 37, 2001, pp. 35–52.
9. Waller, S. T., and A. K. Ziliaskopoulos. On the Online Shortest Path Problem with Limited Arc Cost Dependencies. *Networks*, Vol. 40, 2002, pp. 216–227.
10. Provan, J. S. A Polynomial-Time Algorithm to Find Shortest Paths with Recourse. *Networks*, Vol. 41, 2003, pp. 115–125.
11. Fan, Y., R. Kalaba, and J. Moore. Shortest Paths in Stochastic Networks with Correlated Link Costs. *Computers and Mathematics with Applications*, Vol. 49, 2005, pp. 1549–1564.
12. Gao, S., and I. Chabini. Optimal Routing Policy Problems in Stochastic Time-Dependent Networks. *Transportation Research, Part B*, Vol. 40, No. 2, 2006, pp. 93–122.
13. Frank, H. Shortest Paths in Probabilistic Graphs. *Operations Research*, Vol. 17, 1969, pp. 583–599.
14. Mirchandani, P. B. Shortest Distance and Reliability of Probabilistic Networks. *Computers and Operations Research*, Vol. 3, 1976, pp. 347–355.
15. Sigal, C. E., A. Alan, B. Pritsker, and J. J. Solberg. The Stochastic Shortest Route Problem. *Operations Research*, Vol. 28, 1980, pp. 1122–1129.
16. Loui, R. P. Optimal Paths in Graphs with Stochastic or Multidimensional Weights. *Communications of the ACM*, Vol. 26, 1983, pp. 670–676.
17. Markowitz, H. Portfolio Selection. *Journal of Finance*, Vol. 7, 1952, pp. 77–91.
18. Eiger, A., P. B. Mirchandani, and H. Soroush. Path Preferences and Optimal Paths in Probabilistic Networks. *Transportation Science*, Vol. 19, 1985, pp. 75–84.
19. Murthy, I., and S. Sarkar. A Relaxation-Based Pruning Technique for a Class of Stochastic Shortest Path Problems. *Transportation Science*, Vol. 30, 1996, pp. 220–236.

20. Murthy, I. and S. Sarkar. Stochastic Shortest Path Problems with Piecewise Linear Concave Linear Functions. *Management Science,* Vol. 44, 1998, pp. 125–136.
21. Yu, G., and J. Yang. On the Robust Shortest Path Problem. *Computers and Operations Research,* Vol. 25, 1998, pp. 457–468.
22. Montemani, R., and L. Gambardella. An Exact Algorithm for the Robust Shortest Path Problem with Interval Data. *Computers and Operations Research,* Vol. 31, 2004, pp. 1667–1680.
23. Bertsimas, D., and M. Sim. Robust Discrete Optimization and Network Flow. *Mathematical Programming Series B,* Vol. 98, 2003, pp. 49–71.
24. Miller-Hooks, E. D., and H. S. Mahmassani. Least Possible Time Paths in Stochastic, Time-Varying Networks. *Computers and Operations Research,* Vol. 25, 1998, pp. 1107–1125.
25. Miller-Hooks, E. Optimal Routing in Time-Varying, Stochastic Networks: Algorithms and Implementations. PhD thesis. Department of Civil Engineering, University of Texas, Austin, 1997.
26. Miller-Hooks, E. D., 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, 2003, pp. 67–82.
27. Bard, J., and J. Bennett. Arc Reduction and Path Preference in Stochastic Acyclic Networks. *Management Science,* Vol. 37, No. 2, 1991, pp. 198–215.
28. Nie, Y., and X. Wu. The Shortest Path Problem Considering On-Time Arrival Probability. *Transportation Research, Part B,* Vol. 43, No. 6, 2009, pp. 597–613.
29. Hansen, P. Bicriterion Path Problems. In *Multiple Criteria Decision Making: Theory and Application* (G. Fandel and T. Gal, eds.), Springer-Verlag, New York, 1979, pp. 109–127.
30. Fan, Y., and Y. Nie. Optimal Routing for Maximizing the Travel Time Reliability. *Networks and Spatial Economics,* Vol. 3, No. 3, 2006, pp. 333–344.
31. Nie, Y., and X. Wu. Reliable A Priori Shortest Path Problem with Limited Spatial and Temporal Dependencies. In *Proc., 18th International Symposium on Transportation and Traffic Theory,* in press, 2009.
32. Hadar, J., and W. Russell. Rules for Ordering Uncertain Prospects. *American Economic Review,* Vol. 59, 1969, pp. 25–34.
33. Fan, Y., R. Kalaba, and J. Moore. Arriving on Time. *Journal of Optimization Theory and Applications,* Vol. 127, 2005, pp. 497–513.
34. Henig, M. I. The Shortest Path Problem with Two Objective Functions. *European Journal of Operational Research,* Vol. 25, 1985, pp. 281–291.
35. Bar-Gera, H. Origin-Based Algorithm for the Traffic Assignment Problem. *Transportation Science,* Vol. 36, 2002, pp. 398–417.

The Transportation Network Modeling Committee sponsored publication of this paper.