Speedup Techniques for the Stochastic on-time Arrival Problem

Samitha Samaranayake[1], Sebastien Blandin[2], and Alex Bayen[3]

1 **PhD student, Systems Engineering, University of California Berkeley**
samitha@berkeley.edu
2 **Research Scientist, IBM Research Collaboratory – Singapore**
sblandin@sg.ibm.com
3 **Associate Professor, Electrical Engineering and Computer Science, and Civil and Environmental Engineering, University of California Berkeley**
bayen@berkeley.edu

Abstract

We consider the stochastic on-time arrival (SOTA) routing problem of finding a routing policy that maximizes the probability of reaching a given destination within a pre-specified time budget in a road network with probabilistic link travel-times. The goal of this work is to provide a theoretical understanding of the SOTA problem and present efficient computational techniques to enable the development of practical applications for stochastic routing. We present multiple speedup techniques that include a label-setting algorithm based on the existence of a minimal link travel-time on each road link, advanced convolution methods centered on the Fast Fourier Transform and the idea of zero-delay convolution, and localization techniques for determining an optimal order of policy computation. We describe the algorithms for each speedup technique and analyze their impact on computation time. We also analyze the behavior of the algorithms as a function of the network topology and present numerical results to demonstrate this. Finally, experimental results are provided for the San Francisco Bay Area arterial road network to show how the algorithms would work in an operational setting.

1998 ACM Subject Classification F.2.0 General

Keywords and phrases Stochastic routing, Dynamic programming, Traffic information systems

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.83

1 Introduction

Optimal routing strategies in many practical settings require taking into account some notion of route reliability or travel-time variance in addition to simply considering the expected travel-time of a trip. However, most commercially available routing algorithms do not present this as an option due to the computational time complexity of determining shortest paths with reliability constraints. This work aims at extending the state of the art in computational tractability for a particular type of stochastic shortest path problem known as the *stochastic on-time arrival* (SOTA) problem. In this problem, we wish to determine a routing policy that maximizes the probability of on-time arrival, given an origin destination pair and a desired travel-time budget. Fan et al. [3] formulated the SOTA problem as a stochastic dynamic programming problem and solved it using a standard *successive approximation* (SA) algorithm. In an acyclic network, the SA algorithm converges in a number of steps no greater than the maximum number of links in the optimal path. However, in a network that contains cycles, as is the case with all road networks, there is no finite bound

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 83–97

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

on the maximum number of iterations required for the algorithm to converge [3]. This is due to the fact that the optimal solution can contain loops, as will be explained later. As an alternative, Nie et al. [7] propose a discrete approximation algorithm for the SOTA problem that converges in a finite number of steps and runs in pseudo-polynomial time.

Samaranayake et al. [10] showed how the SOTA problem can be solved exactly in a finite number of steps, even in cyclic networks when there is a minimum realizable link travel-time on every link. As with Fan et al.[3], this algorithm requires computing a continuous-time convolution product, which is one of the main computational challenges of the method. In general, this convolution cannot be solved analytically when routing in general networks, and therefore a discrete approximation scheme is required. The solution presented in [10] allows for batch computation of the convolution product and thus more efficient computation methods than the standard (brute force) discrete time approximation algorithm used in [7]. In this formulation, the order in which the nodes of the graph are considered when solving the underlying dynamic program greatly impacts the running time of the proposed solution. Therefore, an optimal ordering algorithm that determines the best order in which to solve the dynamic program is also proposed.

The contributions of this article are as follows. First we give a concise description of the existing optimization techniques for the SOTA problem, which form the basis for the extensions proposed in this work. We then present a new algorithm that combines the ideas of a minimum realizable travel-time and optimal ordering from [10] and the idea of zero-delay convolution [1, 4] to create a even more efficient solution to the SOTA problem. Complexity results are given for all the optimization techniques presented. We also analyze the computation time of the algorithms as a function of the network topology. The algorithms perform best on networks with long road segments and a limited number of loops. Road networks in general consist of arterial networks with short segments and many loops that are connected via a highway network that contains long segments and fewer loops. The implications of this structure for efficient computation of stochastic shortest paths is discussed. Experimental results are provided both for synthetic networks with varying levels of structural complexity and for the San Francisco Bay Area arterial network using the *Mobile Millennium* [11] traffic information system.

2 Stochastic On-Time Arrival (SOTA) problem

We consider a directed network $G(N, A)$ with $|N| = n$ nodes and $|A| = m$ links. The weight of each link $(i, j) \in A$ is a random variable with probability density function $p_{ij}(\cdot)$ that represents the travel-time on link (i, j). The link travel-time distributions are assumed to be independent[1]. Given a time budget T, an *optimal route* is defined to be a policy that maximizes the probability of arriving at a destination node s within total travel-time of T. A routing policy is an *adaptive* set of instructions that determines the optimal path at each node (intersection in the road network) based on the cumulative travel-time that has already been realized. This is in contrast to a-priori solutions [8, 9] that determine the entire path prior to departure. Given a node $i \in N$ and a time budget t, let $u_i(t)$ denote the probability of reaching the destination node s from a given node i in less than time t when following the optimal policy. At each node i, the traveler should pick the link (i, j) that maximizes the probability of arriving on time at the destination. If j is the next node being visited

[1] See [10] for a formulation that considers localized correlations.

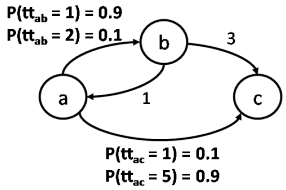
after node i and ω is the time spent on link (i, j), the traveler starting at node i with a time budget t has a time budget of $t - \omega$ to travel from j to the destination[2].

▶ **Definition 1.** The optimal routing policy for the SOTA problem can be formulated as follows:

$$\begin{aligned} u_i(t) &= \max_{j:(i,j)\in A} \int_0^t p_{ij}(\omega)\, u_j(t-\omega)\, d\omega \qquad \forall i \in N,\ i \neq s,\ 0 \leq t \leq T \\ u_s(t) &= 1 \quad 0 \leq t \leq T. \end{aligned} \tag{1}$$

The functions $p_{ij}(\cdot)$ are assumed to be known and can be obtained for example using historical data or real-time traffic information.

One approach to solving this problem would be to use a *successive approximations* (SA) algorithm as in [3], which solves the system of equations (1) repeatedly until convergence and gives an optimal routing policy. At each iteration k, $u_i^k(t)$ gives the probability of reaching the destination node s from a given node i within a travel-time of t, using a path that has no more than k links, when following the policy computed by the algorithm. This is an approximation to the optimal solution that limits the total number of road links in a path to k. The approximation error decreases monotonically with k and the solution eventually reaches an optimal value when k is equal to the number of links in the longest optimal path contained in the policy. However, since an optimal routing policy in a stochastic network can have loops [10] (see Figure 1), the number of iterations required to attain convergence is not known a-priori.



Path	Travel-time	Probability
$\{(a,b),(b,c)\}$	4	0.9
$\{(a,c)\}$	1	0.1
$\{(a,b),(b,a),(a,c)\}$	4	0.01

Figure 1 A simple network with an optimal routing policy that may contain a loop. Links (b, c) and (b, a) have deterministic travel-times of respectively 3 and 1 time units. Link (a, b) has a travel-time of 1 with probability 0.9 and a travel-time of 2 with probability 0.1. Link (a, c) has a travel-time of 5 with probability 0.9 and a travel-time of 1 with probability 0.1. Assume that we wish to find the optimal path from node a to node c with a total travel-time budget of 4. The table presents on-time arrival probabilities for all feasible paths. The optimal solution clearly is to first take link (a, b). However, if the realized travel-time on (a, b) is 2, the only feasible path is to return back to node a and then proceed on link (a, c).

3 Label-setting algorithm

Samaranayake et al. [10] presented an algorithm for finding the optimal solution to the continuous time SOTA problem in a single pass through the time-space domain of the problem when the travel-time on each link is lower bounded by a strictly positive constant, and uniformly bounded on the network. Additionally, the complexity of this algorithm does

[2] In this formulation of the problem, the traveler is not allowed to wait at any of the intermediate nodes. See [10] for the conditions under which travel-time distributions from traffic information systems satisfy the first-in-first-out (FIFO) condition, which implies that the on-time arrival probability can not be improved by waiting at a node.

not depend on the number of links in the optimal path. Let β be the minimum realizable link travel-time across the entire network. β is strictly positive since speeds of vehicles have a finite uniform bound, and the network contains a finite number of links with strictly positive length. Therefore, given $\epsilon \in (0, \beta)$, $\delta = \beta - \epsilon$ is a strictly positive travel-time such that $p_{ij}(t) = 0 \ \forall\, t \leq \delta, (i,j) \in A$. Given a time budget T discretized in intervals of size δ, let $L = \lceil T/\delta \rceil$. The SOTA problem can be solved using Algorithm 2.

Algorithm 2 Single iteration SOTA algorithm [10]

Step 0. Initialization.
$k = 0$
$u_i^k(t) = 0, \ \ \forall i \in N, \ \ i \neq s, \ \ t \in [0, T)$
$u_s^k(t) = 1, \ \ \forall\, t \in [0, T)$

Step 1. Update
For $k = 1, 2, \ldots, L$
$\tau^k = k\delta$
$u_s^k(t) = 1, \ \ \forall\, t \in [0, T)$
$u_i^k(t) = u_i^{k-1}(t), \ \ \forall i \in N, \ \ i \neq s, \ \ t \in [0, \tau^k - \delta]$
$u_i^k(t) = \max_{j:(i,j)\in A} \int_0^t p_{ij}(\omega)\, u_j^{k-1}(t-\omega)\, d\omega, \ \ \forall i \in N, \ \ i \neq s, \ \ t \in (\tau^k - \delta, \tau^k]$

In this formulation of the SOTA problem, the functions $u_i^k(\cdot)$ are computed on $[0, T]$ by increments of size δ. The proposed algorithm relies on the fact that for $t \in (\tau^k - \delta, \tau^k]$, $u_i^k(t)$ can be computed exactly using only $u_j^{k-1}(\cdot)$, $(i,j) \in A$, on $(\tau^k - 2\,\delta, \tau^k - \delta]$, where τ^k is the budget up to which $u_i^k(\cdot)$ is computed at the k^{th} iteration of Step 1. See [10] for proof.

The main computational challenge of this algorithm is calculating the convolution product at each update of $u_i(\cdot)$. It can not be computed analytically since, $u_i(\cdot)$ is the point-wise maximum of the convolution products of the link travel-time distribution $p_{ij}(\cdot)$ with the cumulative distributions of all of its neighboring downstream links $u_j(\cdot)$, $(i,j) \in A$, and the resulting function does not have an analytical expression in general. Since $u_i(\cdot)$ is a continuous monotone increasing function, one solution is to approximate it by a low degree polynomial [2]. However, once again since $u_i(\cdot)$ is a point-wise maximum of multiple functions and its complexity depends on the traffic conditions and the topology of the network, it is in general not well suited for being approximated by a low degree polynomial.

An alternative to computing the convolution by polynomial approximation or other similar methods, is to solve the convolution product via a time discretization of the distributions involved [7], which results in a computational time complexity that is independent of the shape of the optimal cumulative travel-time distributions $u_i(\cdot)$. In the discrete setting, the SOTA problem can be solved using a discretized version of Algorithm 2 [10]. Let Δt ($\leq \delta$) be the length of the discretization interval and T be the time budget. The functions $u_i(\cdot)$ and $p_{ij}(\cdot)$ are now vectors of length $L = \lceil \frac{T}{\Delta t} \rceil$. For notational simplicity, we assume that T is a multiple of Δt. We also assume that the link travel-time distributions are available either as discrete or continuous time distributions. If the link travel-time distributions are discrete and the length of the discretization interval d is not equal to Δt or the distribution is continuous, the probability mass needs to be redistributed to intervals of Δt.

Obtaining the appropriately discretized probability mass functions can be done in time $O(\frac{mT}{\Delta t})$, since there are m links and each link travel-time distribution function is of length $\frac{T}{\Delta t}$. Initializing n vectors (one for each node i) of length $\frac{T}{\Delta t}$ takes $O(\frac{nT}{\Delta t})$ time. As in Al-

gorithm 2 for each link (i,j) the algorithm progressively computes a set of convolutions of increasing length from $x = 1$ to $x = \frac{L\delta}{d} = \frac{T}{\Delta t}$. Therefore, the time complexity of the summation for each link is $O((\frac{T}{\Delta t})^2)$. The assignment $u_i^k(x) = u_i^{k-1}(x)$ can be done in constant time by manipulating pointers instead of a memory copy or by simply having one array for all $u_i(\cdot)$ that keeps getting updated at each iteration of the loop. Since there are m links, the total time complexity is $O(m(\frac{T}{\Delta t})^2)$, which dominates the complexity of the algorithm. The drawback of this method is the numerical discretization error in the representation of the probability density function. A smaller discretization interval leads to a more accurate approximation, but increases the computation time quadratically. Thus, this method still turns out to be computationally intractable for practical settings [10].

A common strategy for speeding up computing convolutions is to use the *Fast Fourier Transform* (FFT). The FFT is an algorithm that computes the convolution of two vectors of length n in $O(n \log n)$ time. Notice however that the proposed algorithm does not compute the entire convolution at once. The computation is required to be done in blocks of length δ for correctness as explained previously. Therefore, L separate convolution products of increasing lengths $\delta, 2\delta, \ldots, L\delta$ need to be computed. FFT based convolution is inefficient in terms of complexity in this setting since successive convolutions recompute results that have already been obtained in previous convolutions. For each link, the time complexity of the sequence of FFTs is $O(\sum_{k=1}^{L} \frac{\delta k}{\Delta t} \log(\frac{\delta k}{\Delta t}))$, where $L = \lceil \frac{T}{\delta} \rceil$. Since there are m links, the total time complexity is:

$$O\left(m \sum_{k=1}^{\lceil \frac{T}{\delta} \rceil} \frac{\delta k}{\Delta t} \log\left(\frac{\delta k}{\Delta t}\right)\right). \tag{2}$$

As $T \to \infty$, the complexity of the FFT based approach for each link is $O\left(\frac{T^2}{\delta \Delta t} \log\left(\frac{T}{\Delta t}\right)\right)$ and asymptotically larger than the run-time of the brute force approach $\sum_{k=1}^{\frac{T}{\Delta t}} k = O\left(\left(\frac{T}{\Delta t}\right)^2\right)$. However, in practice the computation time of the FFT based approach can be smaller than the brute force approach in the time range of interest for most practical applications [10]. Furthermore, the idea of batch computation of the convolution integral can be extended with the localization and optimal ordering techniques presented in the next section to obtain order of magnitude gains in the computation time of the algorithm.

4 Localization and optimal ordering algorithm

As shown in Section 3, the runtime of the FFT based solution is a function of minimum realizable link travel-time, δ, and decreases as the value of δ increases. However, in general, road networks are extremely heterogeneous and can contain a large range of minimum link travel-times from link to link, which can be treated individually to improve the total computation time.

▶ Proposition 1. Let β_{ij} be the minimum realizable travel-time on link (i,j), $\delta_{ij} = \beta_{ij} - \epsilon$ $(0 < \epsilon < \beta_{ij})$ and τ_i be the budget up to which the cumulative distribution function $u_i(\cdot)$ has been computed for node i. For correctness, the following invariant must be satisfied throughout the execution of the algorithm. See [10] for proof.

$$\tau_i \leq \min_j(\tau_j + \delta_{ij}) \ \ \forall (i,j) \in A \tag{3}$$

When computing the cumulative density function $u_i(\cdot)$ using local δ_{ij} values, the growth of τ_i is different across the nodes i, unlike in the previously presented algorithms where τ_i

grows at the constant uniform rate δ. Furthermore, it turns out that when $u_i(\cdot)$ is updated asynchronously using the invariant $\tau_i \leq \min_j(\tau_j + \delta_{ij}),\ (i,j) \in A$, the order in which the nodes are updated can impact the runtime of the algorithm. In addition, we show that the maximum update interval for a node i is actually bounded by the minimum realizable travel-time of the smallest loop the node belongs to, denoted by δ_i, and not its minimum link travel-time δ_{ij}.

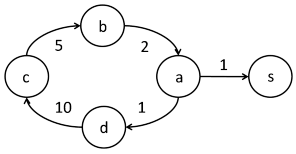


Figure 2 Simple example of a situation where the order in which the dynamic program is solved can have a significant impact of the runtime of the algorithm. The δ_{ij} value for each link is given along the link.

Table 1 τ_i values when computing u_i from values at the previous iteration.

Iter.	a	b	c	d
1	1	2	5	10
2	11	3	7	15
3	16	13	8	17
4	18	18	18	18

Table 2 τ_i values when computing u_i by updating the nodes in the order (a, b, c, d).

Iter.	a	b	c	d
1	1	3	8	18
2	19	21	26	36
3	37	39	44	54
4	55	57	62	72

Table 3 τ_i values when computing u_i by updating the nodes in the order (d, c, b, a).

Iter.	d	c	b	a
1	10	5	2	11
2	15	7	13	16
3	17	18	18	18
4	28	23	20	29

To illustrate how the order in which the nodes are updated impacts the runtime of the SOTA algorithm, consider the network in Figure 2. The value of τ_i at each step and the total computation time of the FFT depends on the order in which the nodes are considered. Table 1 shows the sequence of updates for four iterations when the nodes are updated using the invariant constraint from the previous update iteration. Table 2 shows the sequence of updates when the nodes are considered in the topological order (a, b, c, d). Table 3 shows the sequence of updates when the nodes are considered in the order (d, c, b, a). The highest speedup is achieved when the nodes in the loop are considered in topological order. As seen in Table 2, in that case, the τ_i value for each node i can be incremented by δ_i, the length of the shortest loop node i belongs to, at each step. The optimal order can be determined easily in this simple example, but is non-trivial in complex transportation networks.

Given that the runtime of the SOTA algorithm depends on the update order, we would like to find an optimal ordering that minimizes the runtime of the algorithm. The first step in finding such an optimal ordering is to formalize the runtime of the FFT SOTA algorithm.

▶ **Definition 2.** The computation time of the cumulative density function $u_i(\cdot)$ can be

minimized by finding the ordering that solves the following optimization problem [10].

$$
\begin{aligned}
\text{minimize}_{(\tau_i^{k_i}, K_i)} \quad & \sum_{(i,j)\in A} \sum_{k_i=1}^{K_i} \frac{\tau_i^{k_i}}{\Delta t} \log \frac{\tau_i^{k_i}}{\Delta t} && && (4)\\
\text{subject to} \quad & \tau_i^{k_i} \leq \tau_j^{k_j} + \delta_{ij} && \forall \tau_i^{k_i}, \tau_j^{k_j} \ \ s.t. \ \ (i,j) \in A, \\
& && C(i,k_i) < C(j, k_j + 1) \\
& \tau_r^{K_r} \geq T, \ \ \tau_s^1 \geq T \\
& \tau_i^1 \geq \Delta t && \forall i \in N, i \neq s \\
& \tau_i^{k+1} > \tau_i^k && \forall i \in N
\end{aligned}
$$

where $\tau_i^{k_i}$ is the budget up to which $u_i(\cdot)$ has been computed in the k_i^{th} iteration of computing $u_i(\cdot)$, $C(\cdot,\cdot)$ is an index on the order in which nodes are updated such that $C(i,k_i)$ denotes when node i is updated for the k_i^{th} time and K_i is the total number of iterations required for node i.

Note that the optimal order in which $u_i(\cdot)$ is computed might result in updating some set of nodes multiple times before updating another set of nodes. Samaranayake et al. [10] showed how an algorithm very similar to Dijkstra's shortest path algorithm can be used to find this optimal update order and the size of the updates at each step. The algorithm works by initially considering the source node r and the time budget T to which it needs to be updated, and then recursively updating the set of constraints that need to be satisfied before $u_r(T)$ can be computed. At the first iteration, the source and its terminal value in the algorithm (the budget) are added to a stack χ, and the constraints that are required for updating the source to that value are stored in a heap ψ. At each iteration, the largest value in the heap is extracted and added to the stack, since it is the most constrained node in the current working set. The optimal order of updates (node and value) that computes the cumulative distribution function $u_r(T)$ at the origin r most efficiently is stored in the stack χ at the termination of the algorithm. A more detailed description of the algorithm including the pseudocode and proof of correctness can be found in [10].

5 Efficient convolutions

As explained in section 3, the major computational overhead of the SOTA algorithm is the numerical computation of the convolution products in the dynamic program. While the use of localization with the optimal ordering algorithm, as explained above minimizes the total computation time spent on convolutions, the complexity of the FFT based convolution for each link remains $O\left(\frac{T^2}{\delta_i \Delta t} \log\left(\frac{T}{\Delta t}\right)\right)$, with the only change being the minimum link travel-time δ_{ij} being replaced with the minimum loop travel-time of the upstream node δ_i. The asymptotic complexity as a function of T remains the same since each cumulative distribution function $u_i(\cdot)$ is still recomputed at each update step, as described in section 3. We assume that $\Delta t = 1$ and denote $\delta = \delta_i$ in the rest of this section for notational simplicity.

Gardner [4] proposes an algorithm called *zero-delay convolution* (ZDC) to compute convolutions more efficiently when the input signal is only available in an online fashion, as is the case in our problem. The complexity of convolving two vectors of length n is reduced from $O(n^2 \log n)$ to $O(n \log^2 n)$ when using this technique. ZDC works by constructing the convolution via a series of smaller block convolutions and thereby eliminating the need to recompute sections of the convolution product that have already been computed. Figure 3

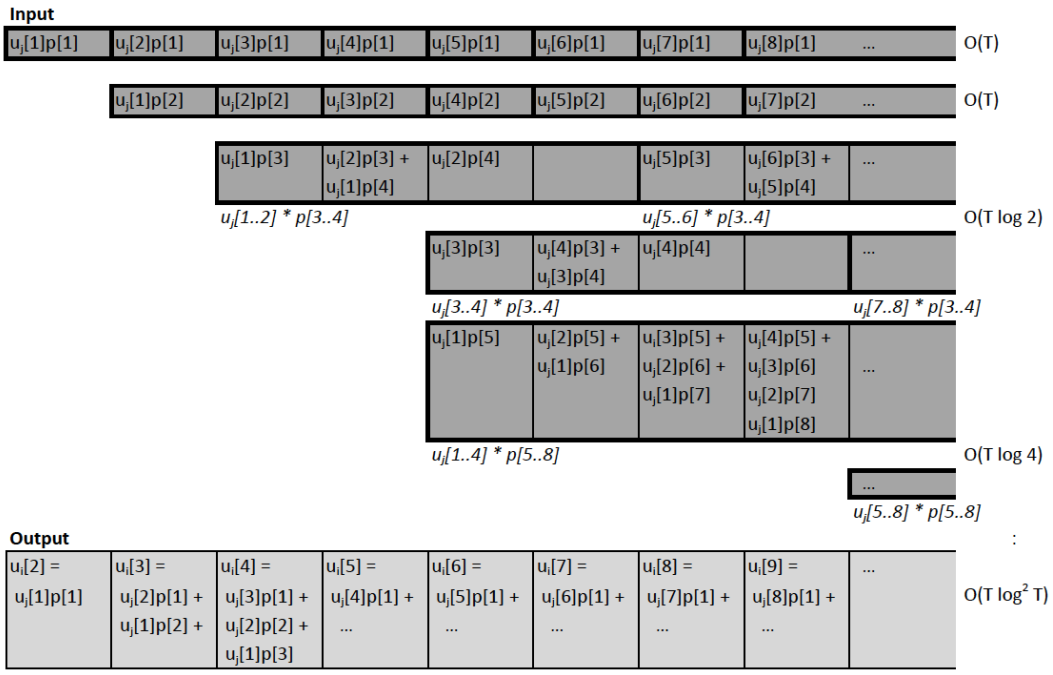


Figure 3 Illustration of the zero-delay convolution algorithm from [1]. The ZDC algorithm computes the convolution one column at a time from left to right. The convolutions are computed in blocks and reassembled to avoid recomputation. The functions $u_i[\cdot]$ and $u_j[\cdot]$ are the discrete cumulative distribution functions for the probability of reaching the destination within some time budget from nodes i and j respectively, where $(i,j) \in A$, and $p[\cdot]$ is shorthand for the probability density function $p_{ij}[\cdot]$. We assume that (i,j) is the only outgoing link from node i. Notice that all the components needed to construct $u_i[k+1]$ are available by the time column k is computed. Some components are computed in advance to exploit the efficiency of block convolutions. The size of the blocks increases exponentially as we proceed through the vectors with the final block having size T. The total computation time is $2T + \sum_{i=1}^{\log T} T \log 2^i = O(T \log^2 T)$.

illustrates the algorithm. Dean [1] shows that ZDC can be applied to the standard SOTA problem to reduce the computational time complexity of the convolutions in each link from $O(T^2 \log T)$ to $O(T \log^2 T)$.

In our setting, ZDC can be combined with the idea of localization to achieve a computational time complexity of $O\left(T\left(\log^2 T - \log^2 \delta\right)\right)$, which can significantly reduce the computation time for networks with large δ values. We call this algorithm δ-multiple ZDC. The process is as follows. First the optimal ordering algorithm is executed to obtain the update steps for all links. Let τ_i^k be the budget up to which $u_i(\cdot)$ has to be calculated to at the k^{th} update for node i. For ease of explanation, without loss of generality we assume that the update interval δ^k is constant over all updates and that both the budget T and update interval δ are powers of two. Without ZDC, $u(\cdot)$ is updated at each step k by convolving two vectors of length $k\delta$ at a cost of $O(k\delta \log(k\delta))$. This sums to a total time complexity of $O\left(\frac{T^2}{\delta} \log T\right)$ as shown in section 3. With δ-multiple ZDC, as with the standard ZDC, the convolution is done in blocks that are reassembled to create the entire convolution. Figure 4 shows a simple example with δ=4. Each block is now twice as large as it was with standard ZDC and the computational time complexity is shown to be $O\left(T\left(\log^2 T - \log^2 4\right)\right)$. More

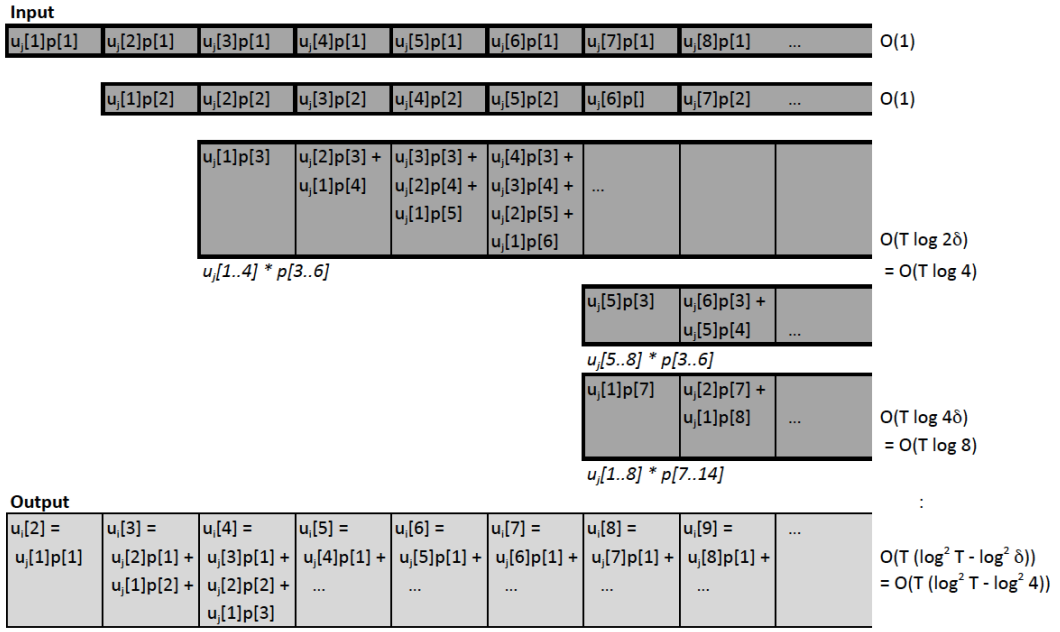


Figure 4 Illustration of the δ-multiple zero-delay convolution algorithm for the SOTA problem. We consider a node i with one downstream link (i, j), where $\delta = 4$ and $\delta_{\text{ij}} = 2$. The functions $u_i[\cdot]$ and $u_j[\cdot]$ are the discrete cumulative distribution functions for the probability of reaching the destination within some time budget from nodes i and j respectively, and $p[\cdot]$ is shorthand for the probability density function $p_{ij}[\cdot]$. The first two rows can be computed in constant time, since $\delta_{\text{ij}} = 2$ implies that $p[1], p[2] = 0$. The rest of the convolution is now computed using block sizes that are multiples of δ making the process more efficient than the standard ZDC. Incorporating localization reduces the computational time complexity from $O\left(T \log^2 T\right)$ to $O\left(T\left(\log^2 T - \log^2 \delta\right)\right)$.

generally, the complexity for each link is:

$$\sum_{i=0}^{\lceil (\log T/\delta) \rceil} O\left(T \log\left(2^i \cdot \delta\right)\right) = O\left(T\left(\log^2 T - \log^2 \delta\right)\right). \tag{5}$$

Since the optimal ordering algorithms pre-computes the maximum update values for each link, the δ-multiple ZDC algorithm can be run with the most efficient δ value for each link, while preserving correctness invariant given in Equation 3.

6 Numerical results

In this section we present numerical results on the performance of the speed-up techniques for the SOTA algorithm presented in the previous sections for two types of networks. First we create a set of synthetic networks to illustrate the relative performance of the base algorithm and its optimizations as a function of the structure of the network. Then we provide some numerical results from implementing the algorithms in a traffic information system for the San Francisco Bay Area. The performance of the algorithm is measured as a function of the total budget T. The algorithms are programmed in Java and executed on an Apple Macbook computer with a 2.4Ghz Intel Core 2 Duo processor and 4GB of RAM. We use the open source Java libraries JTransforms [12] and SSJ [6] for FFT computations and manipulating probability distributions. We consider the following combinations of speed-up

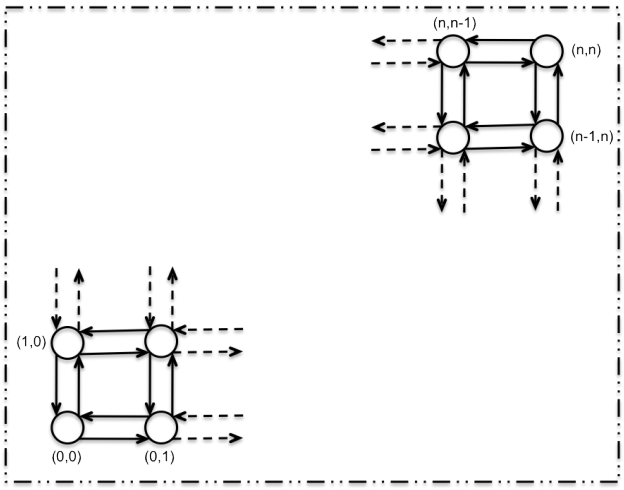


Figure 5 Manhattan Grid with n arcs along the edge of the grid, and minimal link travel-time δ.

techniques[3]:

- **SOTA-Brute force**: convolution as a point-wise shifted product.
- **SOTA-FFT**: convolution using the Fast Fourier Transform algorithm.
- **SOTA-FFT-Opt**: convolution using the FFT algorithm, policy updates according to the optimal ordering algorithm.
- **SOTA-FFT-ZeroDelay**: convolution using the Fast Fourier Transform algorithm in a zero delay framework.
- **SOTA-FFT-ZeroDelay-Opt**: convolution using the Fast Fourier Transform algorithm in a zero delay framework, policy updates according to the optimal ordering algorithm.

6.1 Synthetic network

In this section we analyze the performances of the speed-up techniques proposed on a Manhattan grid (see Figure 5), parameterized by n, the number of arcs on each of the four sides of the grid, δ, the minimal link travel-time, and Δt, the discretization time. We consider the following instantiations of a Manhattan grid:

- Graph A: n=60, $\delta = 5$, $\Delta t = 1$
- Graph B: n=30, $\delta = 10$, $\Delta t = 1$
- Graph C: n=30, $\delta = 20$, $\Delta t = 1$
- Graph D: n=30, $\delta = 5$, $\Delta t = 1$

The link travel-times are chosen as shifted Gamma distributions, with the left support boundary at δ, mean travel-time $\mu = 2\,\delta$, and variance $\sigma = 0.5\,\delta$. The origin is defined as the node with coordinates $(0, 0)$ in the grid and the destination is defined as the node with coordinates (n, n) in the grid. Consequently, on algorithm instantiations for which search pruning is used (implicitly via the optimal ordering algorithm in this case), an inflexion

[3] The successive approximations algorithm from [3] is not considered, since SOTA-Brute force has been shown to outperform it in [7].

point in the runtime can be observed at the budget corresponding to the minimal origin-destination travel-time, corresponding to the fact that the whole graph has been explored by the SOTA policy computation method proposed at this point.

As detailed in the previous section, the runtime of the algorithm depends on the graph size, and on the discretized minimal loop size. In an operational setting, typical nation-wide road networks are composed of two fundamentally different network types, which differ by the inherent structure of their associated graphs, characterized by their minimal graph loop size. Highway networks exhibit large loop sizes, whereas arterial networks are characterized by small loop sizes. Figure 6 illustrates the impact of the network structure over the performances of the proposed speed-up techniques for the SOTA algorithm. For a given budget, and fixed discretized loop size, the runtime on a highway network, with large loop travel-times (Figure 6, right), is significantly reduced compared to the runtime on a arterial network, with small loop travel-times (Figure 6, left). Over hybrid nation-wide networks composed of highway and arterial components, the performance of the algorithm is constrained by the policy computation on arterial networks.

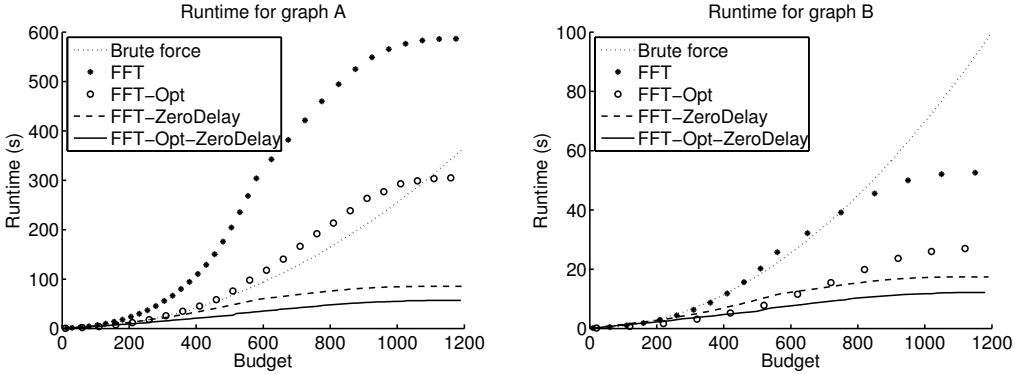


Figure 6 Runtime as a function of budget for different graph structures: Runtime for computing the optimal policy for graph A, left, with $n = 60$, $\delta = 5$, $\Delta t = 1$, and graph B, right, with $n = 30$, $\delta = 10$, $\Delta t = 1$. The brute force method is represented in dotted line, SOTA-FFT using star markers, SOTA-FFT-OPT using circle markers, SOTA-FFT-ZeroDelay in dashed line, and SOTA-FFT-OPT-ZeroDelay in solid line.

Figure 7 illustrates the impact of the network size over the performances of the speed-up techniques. For two graphs with identical network structure, and the larger network (figure 7, left) having travel-times on average four times as long than those of the smaller network (figure 7, right), the benefits of the speed-up techniques are more pronounced in the larger network. The SOTA-FFT-ZeroDelay and SOTA-FFT-OPT-ZeroDelay algorithms scale well with the network size, since the computational time complexity is sub-quadratic in the time budget.

6.2 San Francisco Arterial Network

This section presents experimental results comparing the various versions of the SOTA algorithm on a real network from the San Francisco Bay Area. The algorithms are implemented within the *Mobile Millennium* [11] traffic information system and we test them on the San Francisco arterial sub-network. The network contains 1069 nodes and 2644 links. The travel-time distributions are estimated using the statistical learning algorithm described in [5] using a mixture of real-time and historical probe-generated travel-times.

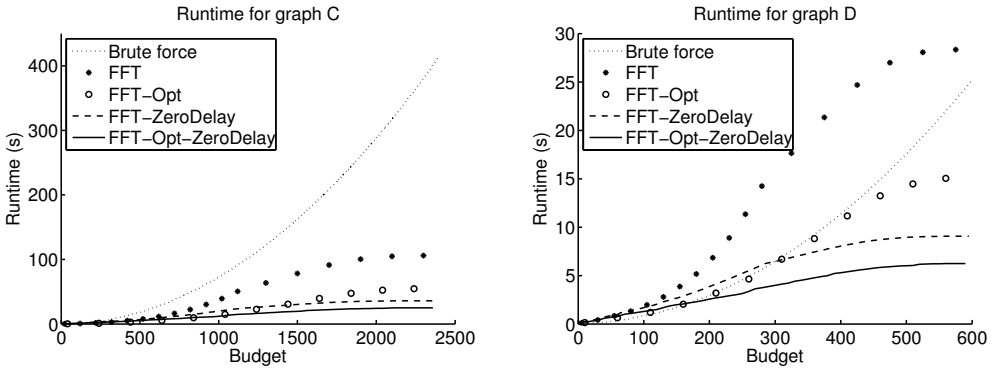


Figure 7 Runtime as a function of budget for different graph size: Runtime for computing the optimal policy for graph C, left, with $n = 30$, $\delta = 20$, $\Delta t = 1$, and for graph D, right, with $n = 30$, $\delta = 5$, $\Delta t = 1$. The brute force method is represented in dotted line, SOTA-FFT using star markers, SOTA-FFT-OPT using circle markers, SOTA-FFT-ZeroDelay in dashed line, and SOTA-FFT-OPT-ZeroDelay in solid line.

Time-varying link travel-time distributions are obtained a-posteriori from the traffic estimation model. The link travel-time distributions are assumed to be independent. We present the actual run-times (in CPU time) for a sample origin-destination (OD) pair (see Figure 8, right), when computing the optimal policy over a range of travel-time budgets.

As illustrated in table 4, the speed-up techniques introduced in this article provide a significant gain in runtime for the SOTA algorithm. The consideration of batch computation via FFT-based convolution (SOTA-FFT), presented in section 3, increases the runtime compared to the brute force method (SOTA-Brute force) due to the inefficiency of computing multiple convolutions products for the same link, however it allows the use of a localization technique (SOTA-FFT-OPT), introduced in section 4, providing an order of magnitude speed-up compared to SOTA-FFT overall, and a factor 2 speed-up, for a budget of 30 minutes compared to SOTA-Brute force. Additionally, the zero-delay convolution method, introduced in section 5, provides an order of magnitude speed-up (SOTA-FFT-ZeroDelay-OPT) compared to the localized algorithm (SOTA-FFT-OPT). Overall, the combination of the localization technique and the zero-delay convolution bring the runtime on a standard laptop from values comparable to the travel budget, to values below the minute for city-level trips, which fall into the practical range for real-time transportation applications.

The three best combinations of the speed-up techniques are also illustrated in Figure 8, left. The impact of the localization technique, which induces a pruning of the graph and leads to policy updates only for vertices that are feasible given the travel budget, is visible in the typical shape of the curves corresponding to SOTA-FFT-OPT and SOTA-FFT-ZeroDelay-OPT, which illustrate that for large budget, the marginal increase in computation time is limited when using the localization technique because fewer additional vertices are feasible. On the other hand the computation time curve for SOTA-FFT-ZeroDelay has a convex shape. The complexity reduction provided by the zero-delay method combined with localization is also clearly visible by comparing the computation times for SOTA-FFT-OPT and SOTA-FFT-ZeroDelay-OPT, which decrease from around 15 minutes to around 1 minute for a budget of 30 minutes.

Table 4 Runtime (in minutes) for different budgets.

Algorithm	Budget 10 minutes	Budget 20 minutes	Budget 30 minutes
SOTA-Brute force	3.3	13.0	29.2
SOTA-FFT	19.1	73.2	154.9
SOTA-FFT-OPT	2.7	9.8	15.0
SOTA-FFT-ZeroDelay	0.8	2.7	5.2
SOTA-FFT-ZeroDelay-OPT	0.3	0.8	1.1

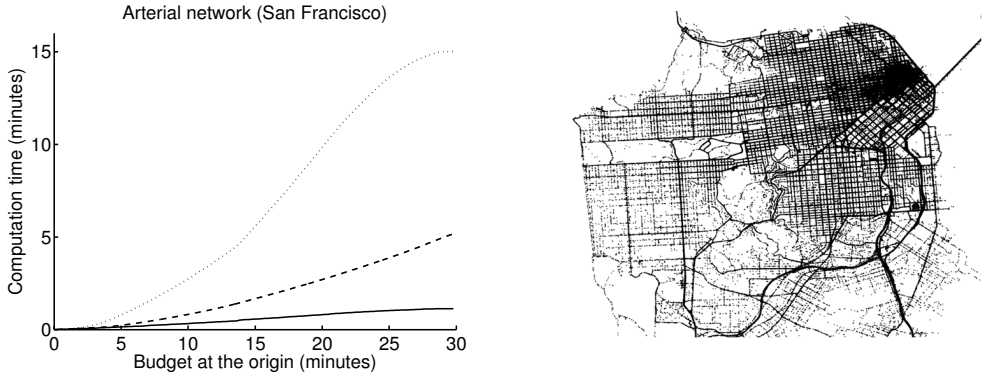


Figure 8 Left: Runtime for computing the optimal policy for a route from the Financial District (Columbus and Kearny) to the Golden Gate Park (Lincoln and 19th) Comparison of run-times (CPU time) for SOTA-FFT-OPT (dotted line), SOTA-FFT-ZeroDelay (dashed line), SOTA-FFT-ZeroDelay-OPT (solid line). The time discretization (Δt) is 0.4 seconds. **Right: Illustration of the San Francisco Arterial network** Cumulated probe data measurements from the San Francisco arterial network for a single day.

7 Conclusions and future work

This article presents a collection of optimization techniques that can be used to improve the tractability of the SOTA problem and move closer to the eventual goal of implementing a real-time stochastic router in an operational setting. All the optimization techniques are based on the existence of an uniform strictly positive minimum link travel-time. This allows us to compute the SOTA solution using a label-setting algorithm instead of a label correcting successive approximations scheme. It also allows for batch computation of the convolution integrals which is a key component of the optimization techniques. It is seen that the heterogeneity of the minimum link travel-times on a network can make the SOTA algorithm very sensitive to the order in which the nodes are treated. An optimal ordering algorithm is then presented to find the update order that minimizes the computational time complexity. Finally, a technique to compute convolutions for streaming signals efficiently, zero-delay convolution (ZDC), is extended to create a new δ-multiple ZDC algorithm that reduces the time complexity of each convolution product to $O(T\left(\log^2 T - \log^2 \delta_i\right)$, where δ_i is the minimum strictly positive loop travel-time for node i. Experimental results are provided to numerically justify the theoretical contributions.

While all the results presented here focus on exact solutions to the SOTA problem, practical routing applications rarely require the problem to be solved exactly. The tractability of the problem has the potential to be improved significantly using approximation algorithms.

Additional improvements could also be archived via heuristic search pruning algorithms and pre-processing methods similar to those used in the deterministic shortest path problem to reduce the computation times by multiple orders of magnitude. Even though we have presented techniques for order of magnitude improvements in solving the exact SOTA problem, further runtime reductions via approximation algorithms and heuristics will hold the key to being able to implement stochastic shortest path algorithms in mainstream vehicle routing systems.

References

1 B.C. Dean. Speeding up stochastic dynamic programming with zero-delay convolution. *Algorithmic Operations Research*, 5(2), 2010.

2 Y.Y. Fan, R.E. Kalaba, and J.E. Moore. Arriving on time. *Journal of Optimization Theory and Applications*, 127(3):497–513, 2005.

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

4 W.G. Gardner. Efficient convolution without input-output delay. *Journal of the Audio Engineering Society*, 43(3):127–136, 1995.

5 R. Herring, A. Hofleitner, and A. Bayen. Estimating arterial traffic conditions using sparse probe data. In *13th International Conference on Intelligent Transportation Systems*, Madeira Island, Portugal, 2010.

6 P. L'Ecuyer. Stochastic simulation in java, `http://www.iro.umontreal.ca/~simardr/ssj/indexe.html`, 2008.

7 Y. Nie and Y. Fan. Arriving-on-time problem. *Transportation Research Record*, pages 193–200, 2006.

8 Y. Nie and X. Wu. Reliable a priori shortest path problem with limited spatial and temporal dependencies. In *International Symposium on Transportation and Traffic Theory, Hong Kong*. Springer, 2009.

9 E. Nikolova. Optimal route planning under uncertainty. In *Proceedings of International Conference on Automated Planning and Scheduling*. Citeseer, 2006.

10 S. Samaranayake, S.Blandin, and A. Bayen. A tractable class of algorithms for reliable routing in stochastic networks. *Transportation Research Part C*, 20(1):199–217, 2011.

11 *Mobile Millennium*. http://traffic.berkeley.edu, 2008.

12 P. Wendykier. Jtransforms, `http://sites.google.com/site/piotrwendykier/software/jtransforms`, 2009.

Revision Notice

This is a revised version of the eponymous paper appeared in the proceedings of ATMOS 2012 (OASIcs, volume 25, `http://www.dagstuhl.de/dagpub/978-3-939897-45-3`, published in September, 2012), in which a a single ratio was incorrectly stated in the paragraph following equation (2) on page 87. This error was fixed and minor changes were made where necessary (e.g. Fig. 3 and Fig. 4).

Dagstuhl Publishing – November 16, 2012.