Precomputation techniques for the stochastic on-time arrival problem

Guillaume Sabran[*] Samitha Samaranayake[†] and Alexandre Bayen[‡]

Abstract

We consider the stochastic on-time arrival (SOTA) problem of finding the optimal routing strategy for reaching a given destination within a pre-specified time budget and provide the first results on using preprocessing techniques for speeding up the query time. We start by identifying some properties of the SOTA problem that limit the types of preprocessing techniques that can be used in this setting, and then define the stochastic variants of two deterministic shortest path preprocessing techniques that can be adapted to the SOTA problem, namely reach and arc-flags. We present the preprocessing and query algorithms for each technique, and also present an extension to the standard reach based preprocessing method that provides additional pruning. Finally, we explain the limitations of this approach due to the inefficiency of the preprocessing phase and present a fast heuristic preprocessing scheme. Numerical results for San Francisco, Luxembourg and a synthetic road network show up to an order of magnitude improvement in the query time for short queries, with even larger gains expected for longer queries.

1 Introduction

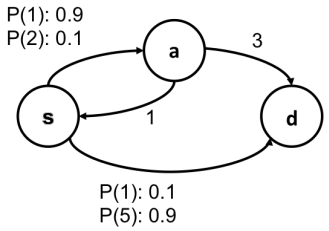
Arriving on time is an important consideration in many practical routing problems, which raises two questions: "When should I leave?" and while en-route "which routing strategy should I follow?" The answers to these questions depend on the nature of the travel times in the network. If the travel times are deterministic, there is a vast literature on deterministic shortest path (SP) algorithms that can be use to solve the problem. However, in practice the link travel times are typically random variables with some probability distribution. Deterministic routing algorithms can still be used if the goal is to minimize the expected travel time of the route, due to linearity of expectation, but this does not account for the variance in the travel times, which is an important consideration in many settings.

Stochastic routing. There are two different approaches that have been developed to solve this problem. The first approach is to find the optimal *a priori* path that maximizes the probability of on-time arrival and is known as the shortest path with on-time arrival reliability (SPOTAR) problem. Nie and Wu [12] proposed a general solution to the problem based on first-order stochastic dominance of paths, but the solution is exponential in the worst case runtime. They also present a pseudo-polynomial algorithm that gives an approximate solution to the problem and performs well in practice. Nikolova et. al. [14] showed how to solve the problem in $n^{\Theta(\log n)}$ time when the link travel time distributions are Gaussian.

The second approach is to find a routing policy that determines the optimal route by selecting the best next direction at each junction, and is known as stochastic on-time arrival (SOTA) problem. Such a strategy can result in different paths based on the realized travel times at intermediate road segments and will provide a success probability that is greater than or equal to that of the SPOTAR path. Fan et al. [4] formulated the SOTA problem as a stochastic dynamic programming problem and solved it using a standard *successive approximation* (SA) algorithm. However, in a network that contains cycles, as is the case with all road networks, there is no finite bound on the maximum number of iterations required for the algorithm to converge. This is due to the fact that the optimal solution can contain loops, as illustrated in figure 1. As an alternative, Nie et al. [11] proposed a discrete approximation algorithm for the SOTA problem which converges in a finite number of steps and runs in pseudo-polynomial time. Samaranayake et al. [15, 16] presented a number of optimization techniques to speed up the computation including a label-setting algorithm based on the existence of a uniform strictly positive minimum link travel time, 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.

Preprocessing. Unfortunately, the current state of the art solutions are still too slow to be implemented in commercial navigation systems. The goal of this

[*]Ecole Polytechnique. guillaume.sabran@polytechnique.org
[†]University of California, Berkeley. samitha@berkeley.edu
[‡]University of California, Berkeley. bayen@berkeley.edu



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

Figure 1: A simple network with an optimal routing policy that may contain a loop. Links (a,d) and (a,s) have deterministic travel times of respectively 3 and 1 time units. Link (s,a) has a travel time of 1 with probability 0.9 and a travel time of 2 with probability 0.1. Link (s,d) 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 s to node d 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 (s,a). However, if the realized travel time on link (s,a) is 2, the only feasible path is to return back to node s and then proceed on link (s,d).

work is to further improve the computation time of the SOTA problem by adapting preprocessing techniques that have been used very successfully in the deterministic SP setting. These techniques include goal directed search methods such as $A^\star$, arc-flags [2, 9] and ALT [7], and algorithms that exploit the hierarchy of road networks such as reach [8], contraction hierarchies [5], and transit-node routing [1]. These algorithms can provide speedups of over three orders of magnitude over Dijkstra's algorithm, and solve networks with 20 million nodes and 50 million edges with sub-millisecond runtimes [1] by exploiting the structure of road networks. Since this structure also exists in the stochastic setting, we would like to try and adapt these preprocessing techniques to the SOTA problem.

Contributions. Our contributions are as follows. This work is to the best of our knowledge the first attempt to use graph preprocessing techniques to speed up the computation of stochastic SP problems, and in particular the SOTA problem. We analyze certain limiting properties of the SOTA formulation that differ from the deterministic SP problem and determine which preprocessing techniques can be adapted to the SOTA problem. We then provide stochastic definitions of the reach and arc-flags methods (two techniques that can be used in the SOTA setting) and describe the algorithms for both preprocessing and query-processing. Finally, we explain the computational inefficiency of the preprocessing phase due to the constraints of the SOTA formulation and introduce some ideas for fast heuristic schemes. An experimental analysis is provided for a synthetic network that represent a Manhattan grid, and the San Francisco and Luxembourg road networks. Numerical results show up to an order of magnitude improvement in the query time for short queries[1], with the expectation of even larger gains for longer queries.

2 Preliminaries

We consider a directed graph $G(V,E)$ that represents the road network. The weight of each link $(i,j) \in E$ 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[2]. Given a time budget T, an *optimal routing policy* is defined to be a policy that maximizes the probability of arriving at a destination node d within a 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 the SPOTAR solution [12, 13] that determines a fixed path prior to departure. Given a node $i \in V$ and a time budget t, let $u_{id}(t)$ denote the probability of reaching the destination node d from a given node i in less than time t when following the optimal policy, and let $n_{id}(t)$ be the optimal next node to visit. 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 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[3].

The optimal routing policy for the SOTA problem can be obtained by solving the following system of equations.

$$u_{id}(t) = \max_{j:(i,j)\in E} \int_0^t p_{ij}(\omega)\, u_{jd}(t-\omega)\, d\omega \qquad \forall i \in V,\ i \neq d,\ 0 \le t \le T \tag{2.1}$$

$$u_{dd}(t) = 1 \qquad 0 \le t \le T$$

[1] The maximum time budget of our queries is limited by the memory limitations of the hardware and Java implementation

[2] See [16] for a formulation that considers local correlations.

[3] In this formulation of the problem, the traveler is not allowed to wait at any of the intermediate nodes. See [16] 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.

$$n_{id}(t) = \arg\max_{j:(i,j)\in E} \int_0^t p_{ij}(\omega) u_{jd}(t-\omega) d\omega \quad \forall i \in V,\ i \neq d,\ 0 \leq t \leq T \tag{2.2}$$

One approach to solving this problem would be to use a *successive approximations* (SA) algorithm as in [4], which solves the system of equations (2.1) repeatedly until convergence and gives an optimal routing policy. Samaranayake et al. [16] presented an algorithm for finding the optimal solution 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. They also presented multiple optimization techniques to speed up the computation [15, 16], but the performance of all of these algorithms is limited by the large search space of the problem. The objective of this work is to further speed up the computation time by reducing the search space of the problem, using stochastic adaptations of the preprocessing techniques that have been extremely effective in the deterministic SP problem for road networks.

2.1 Preprocessing constraints of the SOTA problem. Ideally, we would be able to directly apply the ideas from the deterministic SP problem to the SOTA setting with minimal modifications. However, there are some fundamental differences of the two problems that limit the types of preprocessing techniques that can be used in the SOTA framework. The SOTA solution does not satisfy two important properties that are present in the deterministic SP problem; it cannot be computed in the reverse direction and sub-policy optimality does not hold.

Bidirectional search is not possible. Bidirectional search is a common technique used both in the preprocessing and query stages of fast deterministic routing solutions. For example, Contraction Hierarchies [5] and variants of the arc-flags [9] and reach [6] algorithms rely on the ability to perform bidirectional search. However, speedup techniques that rely on bidirectional search can not be applied to the SOTA problem. As can be seen in equation (2.1), the final and intermediate solutions of the SOTA problem are a function of the remaining time budget, which implies that finding the optimal routing strategy requires this knowledge. When performing a bidirectional search, the reverse search will not have this information.

LEMMA 2.1. (SOLUTION ON REVERSE GRAPH) *Let $s, d \in V$ and T be a time budget. The SOTA problem of reaching d from s within T in G is not equivalent to reaching s from d in the reverse graph with the same time budget.*

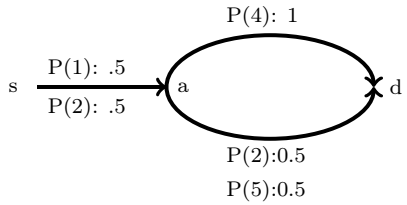


Figure 2: A network where the forward and reverse problems are not equivalent

Proof by contradiction. Figure 2 depicts a network in which we wish to find the SOTA solution for traveling from the source s to the destination d within 5 time units. The probability of reaching d from s within 5 units of time is 0.75 and the probability of reaching s from d within the same time budget in the reverse graph is 0.5. The reason for this difference is the following. In the forward problem, the path decision from a to d is made using the information about the travel time from s to a and knowing the remaining time budget. However, in the reverse problem, the decision on which path to take from d to a must be made without any information on the realized travel time between a and s.

Sub-policy optimality does not hold. Sub-path optimality is another commonly utilized property when solving SP problems. If a destination node d has two incoming links a and c (as in figure 3), then the optimal path from a source node s to d is either the optimal path from $s \to a$ plus (a, d) or the optimal path from $s \to c$ plus (c, d). However, this basic assumption does not hold in the SOTA setting. To be precise, the optimal SOTA policy from s to d can not be constructed using the optimal policies from s to a and s to c. This prevents us from being able to construct stochastic variants of some the most effective preprocessing techniques such as transit nodes [1] and SHARC [2].

DEFINITION 2.1. (OPTIMAL NODE SET) *Let $V_{sd}(T)$ be the set of nodes that span all realizable optimal paths for reaching a destination d from a source s within a time budget T.*

LEMMA 2.2. (SUB-POLICY SUB-OPTIMALITY) *Let $s, d \in V$, T be a time budget and Φ be a vertex separator of $V_{sd}(T)$. The optimal policy from s to d for a time budget of T can not be constructed using only the optimal policies from s to v and v to d for all $v \in \Phi$.*

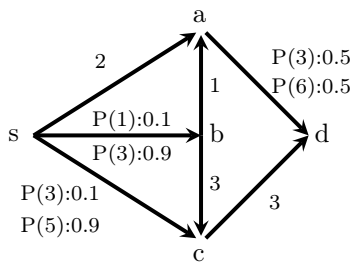


Figure 3: A network where the optimal policy cannot be decomposed

Proof by contradiction. In figure 3, the best path for going from s to a or s to c for any time budget is via the direct edges (s, a) and (s, c). Also, it is clear that it is necessary to travel through either a or c to reach d. However, in this example, the optimal policy for going from s to d with a time budget of 7 is to first go to b and then chose the next edge between a and c based on the remaining time budget.

3 Preprocessing techniques for SOTA

In this section we will focus on two preprocessing techniques that can be adapted to work with the SOTA problem, namely reach and arc-flags. We first define the stochastic variants of these techniques and explain how they can be used for fast query processing in the SOTA problem.

3.1 Reach. The reach [8] of a node is a metric that quantifies the radius of a node's relevance. A node with a small reach value will only belong to shortest paths whose source or destination are close to the node, while a node with a large reach may belong to shortest paths involving sources and destinations that are far from it. We adapt the notion of reach to the stochastic setting of the SOTA problem and present a variant of the reach definition that allows for better pruning.

Definition 3.1. (Stochastic reach) *Let m be a metric and $m(i, j)$ the minimal distance between i and j for this metric. For a node i and some time budget t, we define $\Psi_i(t)$ to be the set of source-destination pairs that contain i in their optimal policy for some time budget smaller than or equal to t.*

$$\Psi_i(t) = \{(s, d) \in V^2 : \exists t' \leq t, i \in V_{sd}(t')\}$$

We define the stochastic reach of a node i for a time budget t as:

$$r(i, t) := \max_{(s,d)\in\Psi_i(t)} \min(m(s, i), m(i, d))$$

The reach of a node can be used to speed up SOTA queries by pruning the graph as described below prior to processing a SOTA query.

Lemma 3.1. (Graph pruning with reach) *Let (s, d) be a source-destination pair and T be the time budget for reaching the destination. Let i be a node and $t \geq T$ a budget for which the reach $r(i, t)$ has been precomputed. Node i can be pruned from the graph without changing the optimal solution if $r(i, t) < \min(m(s, i), m(i, d))$.*

Proof. If node i belongs to the optimal node set for source-destination pair (s, d) with a time budget T, i.e. $i \in V_{sd}(\mathrm{T})$, then $(s, d) \in \Psi_i(t), t > T$ and $r(i, t) \geq \min(m(s, i), m(i, d))$ by definition 3.1. Therefore, if $r(i, t) < \min(m(s, i), m(i, d))$, node i is not on any optimal path for the source-destination pair (s, d) with a time budget T and can be pruned.

Partition-based reach. One drawback of reach pruning is that the pruned graph could contain a large number of false positives. This is to be expected because the reach metric is computed over the set of all source-destination (s, d) pairs in the network. However, we can improve the precision of the reach metric by computing multiple reach values for each node that are conditioned on some information about the (s, d) pair with respect to which the reach is being computed. One such method is to divide the graph into partitions and compute an individual reach value for each partition. We partition all possible source-destination (s, d) pairs into several clusters with respect to the node for which we are computing the reach, and compute the reach value corresponding to each of these clusters. In the pruning phase, we first find the cluster that the source-destination pair belongs to and look up the corresponding reach value. This leads to more precise reach values and improves the pruning ability at the expense of an additional memory requirement. There is a trade off between the precision of the reach and the memory used; the two extrema been the regular reach and computing the reach for every possible source-destination pair. We first provide an abstract definition of partition-based reach and then present a specific partitioning scheme.

Definition 3.2. (Partition-based reach) *Let i be a node. Let S be an arbitrary function such that $S(i)$ is a partition[4] of V^2. For notational simplicity let $S_i = S(i)$. For (s, d) in V^2, let $S_i^{(s,d)}$ be the unique*

[4]Defined as: $\forall P \in S_i, P \neq \emptyset$; $\forall Q \neq P \in S_i, P \cap Q = \emptyset$; $\bigcup_{P\in S_i} P = V^2$.

set of the partition S_i such that $(s,d) \in S_i^{(s,d)}$. Then $\Psi_i(t) \cap S_i^{(s,d)}$ is the set of source-destination pairs that contain i in some optimal policy with a time budget smaller than or equal to t, and which are in the same cluster of S_i as (s,d).

The partition-based reach $r_S(i,t)$ on S is defined as:

$$r_S^{(s,d)}(i,t) = \max_{(s',d') \in \Psi_i(t) \cap S_i^{(s,d)}} \min(m(s',i), m(i,d'))$$

It is easy to see that $r(i,t) = \max_{(s,d) \in V^2} r_S^{(s,d)}(i,t)$

LEMMA 3.2. (PARTITION-BASED REACH PRUNING) *Let (s,d) be a source-destination pair and T be the time budget to reach the destination. Let i be a node and $t \geq T$ a budget for which the reach for the graph has been precomputed. If $r_S^{(s,d)}(i,t) < \min(m(s,i), m(i,d))$, i can be pruned from the graph without changing the optimal solution.*

Proof. If node i belongs to the optimal node set for source-destination pair (s,d) for a time budget T, i.e. $i \in V_{sd}(\mathrm{T})$, then $(s,d) \in \Psi_i(t), t \geq T$. As $(s,d) \in S_i^{(s,d)}$, $r_S^{(s,d)}(i,t) \geq \min(m(s,i), m(i,d))$ by definition 3.2. Therefore, if $r_S^{(s,d)}(i,t) < \min(m(s,i), m(i,d))$, node i is not on any optimal path for the source-destination pair (s,d) with a time budget t and can be pruned.

Example: Directed reach. The original reach definition [8] does not take into account the position of the source and the destination relative to the candidate node to be pruned. If both source and destination are in the same direction from a node i, it is unlikely that this node i will be used in any optimal path as it requires moving away from the destination. This is the motivation for directed reach.

DEFINITION 3.3. (DIRECTED REACH) *Let n be an integer that specifies the number of node sets in the partition and π denote the mathematical constant pi.*

$$I_k := \begin{cases} [\frac{k-1}{n}\pi, \frac{k}{n}\pi[& \text{if } k \in [\![1, n-1]\!] \\ [\frac{n-1}{n}\pi, \pi] & \text{if } k = n \end{cases}$$

We define S_i as the function:

$$i \to \{(s,d) : \widehat{sid} \in I_k\}_{k \in [\![1,n]\!]}$$

where $\widehat{sid}$ is the non-oriented angle between s, i and d.

In directed reach, the source-destination pairs that are likely to contain i in their optimal node set (those with $\widehat{sid}$ close to π) are assigned to the same clusters. The benefits of partition-based reach are validated experimentally in the results section.

Metric. The metric m used when computing the reach of a node does not impact the correctness of the solution, but can influence the quality of the resulting pruning. In our experiments, we compared the average travel time and the minimal travel time metrics on random queries. Experimentation showed that the average travel time generally provides better results, but there could be other metrics that we have not tested that perform better.

3.2 Arc-flags. This method [9] is another well-known query speedup technique used in the deterministic SP problem. The idea is as follows: the graph G is divided into a set of regions R which is a partition of the nodes V. Each edge has an associated vector of booleans (with one value for each region) where each boolean is true if the edge is used by at least one SP ending in the corresponding region. During the query phase, prior to computing the shortest path, any edge that do not have the boolean corresponding to the region that the destination belongs is pruned from the graph. Arc-flags also has the nice property of being able to dynamically update the precomputed data [3]. We adapt arc-flags to the SOTA problem in the same way as we did for reach, by replacing the notion of *belongs to a shortest path* by *might be used by an optimal policy.*

DEFINITION 3.4. (OPTIMAL EDGE SET) *Let $E_{sd}(T)$ be the set of edges that span all realizable optimal paths for reaching a destination d from a source s within a time budget T.*

DEFINITION 3.5. (STOCHASTIC ARC-FLAGS) *For a node d and some time t, we define $\Gamma_d(t)$ to be the set of edges that belong to some path of the optimal policy for traveling from any source s to the destination d with a time budget less than or equal to t.*

$$\Gamma_d(t) = \{e \in E : e \in E_{sd}(t'), t' \leq t, s \in V\}$$

We define the arc-flag of an edge e for a time budget t and a region $r \in R$ as:

$$AF(e,t,r) := \begin{cases} \textit{TRUE if } e \in \bigcup_{d \in r} \Gamma_d(t) \\ \textit{FALSE otherwise} \end{cases}$$

Arc-flags can be used to speed up SOTA queries by pruning the graph as described below prior to processing a SOTA query.

LEMMA 3.3. (QUERY WITH THE ARC-FLAGS) *Let (s,d) be a source-destination pair, r the region d belongs to and T the time budget for reaching d. Let e be an edge and $t \geq T$ a budget for which the arc-flags*

has been precomputed. If $AF(e,t,r)$ is false, e can be pruned from the graph without changing the optimal solution.

Proof. If e belongs to some path of the optimal policy for the source-destination pair (s,d), i.e. $e \in E_{sd}(\mathrm{T})$, then we have $e \in \Gamma_d(t)$ by definition 3.5. Furthermore, since r is the region that the destination d belongs to, it also follows that $AF(e,t,r)$ is true.

3.3 Computing the reach and arc-flags. One of the major limitations of this approach is the large computation time required to calculate the stochastic reach and arc-flags of the network. In the deterministic SP context, this limitation can be overcome by exploiting the property of sub-path optimality to come up with efficient algorithms for computing these metrics. For instance in arc-flags, it is easy to see that one only has to consider the boundary of a region as possible destinations nodes, since the optimal path to any node within the region from a node outside the region will include some optimal path to the boundary of the region. However, as we have seen in section 2.1, this does not work in the SOTA setting. Similarly, the reach metric can also be computed efficiently using a hierarchical method that takes advantage of sub-path optimality, as shown in [8]. Unfortunately, to this point, we have not been able to identify an efficient algorithm for computing the stochastic reach and arc-flags. We are exploring the possibility of upper bounds for the reach metric similar to what is done in the original article by Gutman [8].

Our current approach is to compute the reach and arc-flags in a brute force manner by running a SOTA search for all possible destinations in the graph. Table 4 in the results section shows that this approach is still tractable for some urban scale networks. It is important to note that the computation for each destination is an independent problem and can be done in parallel.

Computing reach. A high level overview of the process is given in Algorithm 3.1. For each destination d, we compute the optimal policy from all sources s to d for the time budget T. The SOTA policy for all sources can be computed simultaneously [16]. Then, for all sources s, we determine the nodes $i \in \bigcup_{t \le T} V_{sd}(t)$ that belong to some optimal path from s to d and update their reach. Determining the set $\bigcup_{t \le T} V_{sd}(t)$ is not trivial since it requires considering all possible realizations of the SOTA policy. We use a efficient priority queue based search with no re-computation of paths, but finding the set $\bigcup_{t \le T} V_{sd}(t)$ and updating the reach from all the sources can take up to twenty times longer than finding the optimal policy. Furthermore, the value $m(s,i)$ might need to be computed multiple times when considering different destinations, but keeping these values in memory for all (s,i) requires too large of a memory footprint. Therefore, we recompute these values for each destination as needed. Fortunately, $m(s,i)$ can be computed efficiently in the sub-graph induced by the nodes $v \in \bigcup_{t \le T} V_{sd}(t)$, which is much smaller than G. This approximation will lead to an upper bound for the reach, but this does not impact the optimality of the solution since it is an upper bound. This bound is usually quite tight since the shortest path is usually close to some path in the SOTA policy.

ALGORITHM 3.1. (REACH COMPUTATION)
input: a graph G and a time budget T
output: the reach $r(\cdot,\mathrm{T})$
initialization: $\forall i \in V$, $r(i,\mathrm{T}) = 0$
for $d \in V$ **do** ▷ can be computed in parallel
  compute the optimal policy with budget T; $\forall s \neq d \in V$
  for $i \in V$ **do**
    compute $m(i,d)$
  for $s \in V$ **do** ▷ can be computed in parallel
    compute $\bigcup_{t \le T} V_{sd}(t)$
    for $i \in \bigcup_{t \le T} V_{sd}(t)$ **do**
      compute $m(s,i)$
      set $r(i,\mathrm{T}) = \max(r(i,\mathrm{T}), \min(m(s,i), m(i,d)))$
return r

Computing arc-flags. Computing the arc-flags of the network is much a more straightforward process, since the arc-flags do not depend on the source. A high level overview of the process is given in Algorithm 3.2. Once again first the optimal policy for each destination is computed for the maximum time budget of interest. Then for each destination the optimal edge set $E_{sd}(t)$ is computed and all the edges in this set are marked as true for the region that the destination belongs to. To optimize the preprocessing, the arc-flags and reach can be computed simultaneously, since they both use the SOTA solution for each destination d and this computation can be shared.

ALGORITHM 3.2. (ARC-FLAGS COMPUTATION)
input: a graph G, a partition of the edges R and a time budget T
output: the arc-flags $AF(\cdot,\mathrm{T},\cdot)$
initialization: $AF(e,\mathrm{T},r) = \mathrm{FALSE}$, $\forall (e,r) \in E \times R$
for $d \in V$ **do** ▷ can be computed in parallel
  compute the optimal policy with budget T; $\forall s \neq d \in V$
  compute $\bigcup_{t \le T, s \in V} E_{sd}(t)$
  for $e \in \bigcup_{t \le T, s \in V} E_{sd}(t)$ **do**
    set $AF(e,\mathrm{T},r(d)) = TRUE$
return AF

4 Experimental results

Test instances. We use three different networks to test our algorithms: a San Francisco arterial network[5] (SF) with 2450 nodes and 6151 edges, a Luxembourg network with 30674 nodes and 72492 edges and a synthetic network with 7921 nodes and 31328 edges. The synthetic network is a 89x89 Manhattan grid with 4 levels of roads, where the speed limits are 40, 60, 80 and 120 kmph, and the size of the network is 40×40 km. The travel time distributions for the SF network is derived from real world observations [10]. The travel time distributions for the Luxembourg network are created artificially using the speed limits as a baseline, as actual travel time information is not available. The distributions for the synthetic network are also generated using the same strategy. All distributions are a mixture of Gaussians corresponding to different traffic modes like slow or fast where all the weight over the speed limit has been moved to the minimal travel time.

Environment. The precomputation was done on 18×1.9GHz AMD Opteron(tm) 6168 processors with 30Gb of shared memory, and the queries where were performed on an Intel Core i7 Q740 with 8×1.73GHz cores and 4Gb of memory. All the code used for the experiments was written in Java 1.6. For each experiment, we randomly picked 200 source-destination pairs with a positive probability of arriving of time. The time discretization of the probability distributions for all experiments is one second.

Technique	Time budget (s)			
	300	500	800	1000
reach (RH)	1.2	1.5	1.4	1.3
directed-RH (DRH)	2.4	3.2	2.9	2.4
arc-flags (AF) 10x10	3.0	5.6	4.6	3.5
DRH & AF 10x10	4.8	8.8	7.2	5.4
baseline runtime (ms)	12	115	579	1554

Table 1: Relative speedups over no preprocessing (San Francisco)

Speedup. Tables 1, 2 and 3 present the average speedups achieved using the preprocessing methods described above compared to computing the results with no preprocessing. The number associated with the arc-flags is the number of regions used. The general speedups achieved are fairly consistent across all the networks. As expected, the directed-reach (DRH) performs better than the regular reach (RH). Also, the performance of the arc-flag (AF) algorithms improves as we increase the number of regions used. Finally, we see that combining reach and arc-flags provides the best results. Figure 4 shows a visualization of how the four different preprocessing algorithms reduce the query-time search space of the problem. One important observation is that the speedups achieved using the preprocessing techniques increases with the time budget of the query. The decrease in the San Francisco network for large budgets is due to the boundary effects of the smaller graph. This is important because the total computation time increases with the budget and the efficiency of the SOTA algorithm must scale well with the time budget.

Technique	Time budget (s)			
	500	1000	1500	2000
reach (RH)	1.5	1.7	1.9	2.2
directed-RH (DRH)	2.0	2.2	2.4	2.8
arc-flags (AF) 20x20	1.5	2.3	3.3	4.8
DRH & AF 20x20	2.5	3.8	5.2	7.2
baseline runtime (ms)	12	283	883	1369

Table 2: Relative speedups over no preprocessing (Luxembourg)

Technique	Time budget (s)			
	500	1000	1500	2000
reach (RH)	1.2	1.3	1.4	1.4
directed-RH (DRH)	1.7	2.0	2.1	2.4
arc-flags (AF) 10x10	2.0	3.0	3.2	3.6
arc-flags (AF) 20x20	3.3	5.1	5.8	5.5
DRH & AF 10x10	2.7	4.6	4.9	5.4
DRH & AF 20x20	3.9	6.8	7.5	8.1
baseline runtime (ms)	23	315	1293	4843

Table 3: Relative speedups over no preprocessing (synthetic network)

The stochastic reach and arc-flags algorithms are generally less efficient than their deterministic counterparts in term of proportional speedup. The first explanation we can give is that the set $V_{sd}(\mathrm{T})$ contains more nodes than a specific shortest path, 50% more on average in our experiments. This means that the individual reach values of nodes are likely to be higher in the stochastic setting and that more edges are likely to be labeled as true in stochastic arc-flags. The stochastic reach and arc-flags are also functions of the time budget being considered, which makes it hard to give meaningful comparisons with the deterministic versions. The second explanation is that we have not performed queries for large time budgets due to computational resource limitations. We have limited the range of our preprocessing to trips of 2000 seconds. The best speedups

[5] We artificially modify the original network by adding two fast roads (highways) in the South/North and East/West directions, since the original network does not contain highways and thus has poor hierarchy.

Figure 4: pruning of the San Francisco network for some source-destination pair. From left to right the pruning achieved using reach, directed-reach, arc-flags and the combination of reach with arc-flags. The colored nodes are the nodes that belong in the optimal policy and the color denotes the probability of the node being used, where blue indicates a high probability and red indicates a low probability. The source is in the bottom of the graph.

Network	Speedup technique						
	reach	arc-flags		heuristic arc-flags			
				10x10		20x20	
	1000	1000	2000	1000	2000	1000	2000
SF	8 497	456	1 556	230	711	317	1 133
Synthetic	18 305	1 303	12 099	584	4 420	860	7 799
Luxembourg	45 643	7 907	75 898	316	5 325	579	8 317

Table 4: Precomputation time in seconds for reach, arc-flags and heuristic arc-flags. Results are presented for maximum time budgets of 1000 and 2000 seconds.

achieved in deterministic road networks are obtained for longer queries. For instance the deterministic reach speedups reported in [8] show that the speedup increases with the length of the path from 4.5 to 19.2 on average for road lengths of 26 and 56 km. Similarly, the significant speedups for the basic arc-flags algorithm were obtained in the German countrywide network where the average trip is much longer [9]. We expect the speedups of the stochastic variants to also increase as we consider longer queries. One of the immediate next steps is to re-implement our algorithms in a programming language with better memory management features and to gain access to better hardware resources to run the experiments on larger networks for larger time budgets.

4.1 Heuristic precomputation. As discussed in section 3.3, preprocessing the graph using stochastic reach and arc-flags is very inefficient due to the additional constraints of the SOTA problem. However, it is possible to efficiently compute heuristics of the reach and arc-flags that are close to optimal values in practice. One such approach is the compute the arc-flags by only considering destinations that are on the boundary of each region, as is done in the deterministic case. This cuts the total number of nodes that need to be considered by a considerable factor (specially when the regions are large and the number of nodes per region is large).

When computing the heuristic arc-flags for the synthetic network we noticed a 0.4% reduction in the total number of arc-flags, i.e. a 0.4% false negative rate. When computing the SOTA solution for 1000 random queries we found 8 non-optimal solutions, and the largest deviation in the probability of arriving on time was 10^{-5}. The results for the SF and Luxembourg networks were similar. Such approaches are especially promising for commercial applications where such minor deviations from the optimal solution are negligible and the measurement error in the probability distributions dominates the error.

The heuristic arc-flags can also be used to compute heuristic reach values as follows. First run Algorithm 3.1 and 3.2 using only destination nodes that are on the boundary of the regions. Then run Algorithm 3.1 from all the other destinations using the heuristic stochastic arc-flags to speed up the optimal policy computation.

5 Conclusions and future work

We present what is to the best of our knowledge the first results on preprocessing techniques for the stochastic on-time arrival (SOTA) problem. We discuss the difficulties in applying the commonly used preprocessing techniques from deterministic shortest paths to the stochastic setting, and identify two techniques (reach and arc-flags) that can be adapted to the SOTA problem. We also present an extension of reach that enables more aggressive pruning of the search space at the cost of some additional memory. The main limitation of this work is the inability to perform the preprocessing in a

computationally efficient manner, making the technique intractable for large networks with large time budgets. However, we discuss the potential for faster precomputation using efficient heuristic schemes, and are experimenting with a number of such techniques. Experimental results show that the preprocessing methods can provide up to a order of magnitude improvement in runtime for the networks we have considered and time budgets on the order of 2000 seconds. We are confident that further refinements to the preprocessing schemes, such as more effective reach partitioning schemes and better strategies selecting regions for arc-flags, will provide further gains. In addition, we plan on reimplementing our algorithms to be more memory efficient and evaluate them on more powerful hardware systems to understand the behavior of the algorithms on larger networks for larger time budgets.

Acknowledgements

The authors would like to thank Timothy Hunter for providing the travel time distributions for the San Francisco network using the path inference filter [10], and Moritz Kobitzsch for sharing sanitized versions of the San Francisco and Luxembourg road networks.

References

[1] Holger Bast, Stefan Funke, and Domagoj Matijevic. Transit—ultrafast shortest-path queries with linear-time preprocessing. *9th DIMACS Implementation Challenge*, 2006.

[2] Reinhard Bauer and Daniel Delling. Sharc: Fast and robust unidirectional routing. *Journal of Experimental Algorithmics (JEA)*, 14:4, 2009.

[3] Gianlorenzo D'Angelo, Daniele Frigioni, and Camillo Vitale. Dynamic arc-flags in road networks. In *Experimental Algorithms*, pages 88–99. Springer, 2011.

[4] Yueyue Fan and Yu Nie. Optimal routing for maximizing the travel time reliability. *Networks and Spatial Economics*, 6(3-4):333–344, 2006.

[5] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Experimental Algorithms*, pages 319–333. Springer, 2008.

[6] Andrew V Goldberg, Haim Kaplan, and Renato F Werneck. Reach for A*: Efficient point-to-point shortest path algorithms. In *ALENEX*, volume 6, pages 129–143, 2006.

[7] Andrew V Goldberg and Renato Fonseca F Werneck. Computing point-to-point shortest paths from external memory. In *ALENEX/ANALCO*, pages 26–40, 2005.

[8] Ronald J Gutman. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *ALENEX/ANALC*, pages 100–111, 2004.

[9] Moritz Hilger, Ekkehard Köhler, Rolf H Möhring, and Heiko Schilling. Fast point-to-point shortest path computations with arc-flags. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, 74:41–72, 2009.

[10] Timothy Hunter, Pieter Abbeel, and Alexandre M. Bayen. The path inference filter: Model-based low-latency map matching of probe vehicle data. In *Algorithmic Foundations of Robotics X*, Springer Tracts in Advanced Robotics, pages 591:1–607:17, Heidelberg, Germany, 2012. Springer-Verlag.

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

[12] Yu (Marco) Nie and Xing Wu. Shortest path problem considering on-time arrival probability. *Transportation Research Part B: Methodological*, 43(6):597 – 613, 2009.

[13] Evdokia Nikolova, Matthew Brand, and David R Karger. Optimal route planning under uncertainty. In *ICAPS*, volume 6, pages 131–141, 2006.

[14] Evdokia Nikolova, Jonathan A Kelner, Matthew Brand, and Michael Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *Algorithms–ESA 2006*, pages 552–563. Springer, 2006.

[15] Samitha Samaranayake, Sebastien Blandin, and Alexandre M Bayen. Speedup techniques for the stochastic on-time arrival problem. In *12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, pages 83–96, 2012.

[16] Samitha Samaranayake, Sebastien Blandin, and Alexandre M Bayen. A tractable class of algorithms for reliable routing in stochastic networks. *Transportation Research Part C: Emerging Technologies*, 20(1):199–217, 2012.