Practical Route Planning Under Delay Uncertainty: Stochastic Shortest Path Queries

Sejoon Lim*, Christian Sommer*, Evdokia Nikolova†, Daniela Rus*
*Computer Science and Artificial Intelligence Lab, MIT
Email: {sjlim, csom, rus}@csail.mit.edu
†Department of Computer Science and Engineering, Texas A&M University
Email: nikolova@tamu.edu

***Abstract*—We describe an algorithm for stochastic path planning and applications to route planning in the presence of traffic delays. We improve on the prior state of the art by designing, analyzing, implementing, and evaluating data structures that answer approximate stochastic shortest-path queries. For example, our data structures can be used to efficiently compute paths that maximize the probability of arriving at a destination before a given time deadline.**

Our main theoretical result is an algorithm that, given a directed planar network with edge lengths characterized by expected travel time and variance, pre-computes a data structure in quasi-linear time such that approximate stochastic shortest-path queries can be answered in poly-logarithmic time (actual worst-case bounds depend on the probabilistic model).

Our main experimental results are two-fold: *(i)* we provide methods to extract travel-time distributions from a large set of heterogenous GPS traces and we build a stochastic model of an entire city, and *(ii)* we adapt our algorithms to work for real-world road networks, we provide an efficient implementation, and we evaluate the performance of our method for the model of the aforementioned city.

Fig. 1. Path query example for maximizing the probability of reaching the destination within a user-specified deadline. Our system finds such a path within milliseconds for a city-sized network. When a user at origin 'O' wants to get to destination 'D' within 30 minutes between 5pm and 6pm on a weekday, the blue path offers the best chance of reaching the destination within the deadline. However, when the deadline changes to 20 minutes, the blue path has less than a 50% chance for making the deadline. In this case, the red path is the best route, offering a 88.5% chance for on-time arrival.

I. Introduction

We present and evaluate a new algorithm for planning the path of vehicles on roadways in the presence of uncertainty in traffic delays. We build on prior work on stochastic path planning [15, 19, 20], which introduces stochastic path planning algorithms in this model. This prior work presents novel provably correct algorithms that are well suited for planning the path of a vehicle on small-scale graphs; however, the computational complexity of this prior suite of stochastic path planning algorithms makes these algorithms impractical to use for real-time path planning on city-scale road networks.

The stochastic path planning model incorporates traffic history in the form of probability distributions of delays observed on actual road segments. The probability distributions capture the historical delay observations that are assigned as weights on the edges of the road network. The stochastic path planning algorithm minimizes a user-specified cost function of the delay distribution. Current route planning methods can produce routes with shortest *expected* time (based on historical traffic data) and some recent commercial systems, using almost-real-time data based on cell-phone locations [22, 26], can also compute "smartest" routes in dynamic traffic scenarios. The algorithm in this paper improves on and extends previous work by enabling fast routes guaranteed to maximize the likelihood of arriving at a destination within a specified deadline in city-scale route graphs. This work provides a planning system that can be used by autonomous as well as human-driven vehicles for traffic routing applications that meet travel goals such as "when should you leave, and what path should you take, to reach the destination by the given deadline with high probability?" Preserving the inherently non-deterministic nature of actual traffic, we work with accurate probabilistic models of traffic-intense areas — in our work, we actually build a probabilistic model of an entire city based on a large set of heterogenous GPS traces.

Our Contributions: The algorithm in this paper improves the state of the art on stochastic path planning [18, 19, 20] and its application to traffic routing [15]. The running time of the state-of-the-art stochastic-shortest-path algorithm is approximately quadratic in the number of nodes, rendering it too slow for real-time performance on city-scale networks. We improve upon their running time by using efficient data structures. In contrast to previous work [18], we provide a two-phase algorithm, consisting of preprocessing and query algorithms, better suited for practical implementations. Our preprocessing algorithm runs in quasi-linear time and it does *not* depend on the origin-destination pair or on the deadline, which is the main theoretical challenge. Our query algorithm

runs in *sublinear* time (whereas the algorithms in [15] run in polynomial time). The running time is *polylogarithmic*, offering exponential speedups over previous algorithms and allowing for almost instantaneous response times (see Lemma 1 and Theorem 3 for details).

We have implemented the algorithms described in this paper and packaged them as a route planning system. We evaluated the system using the road map of an entire city. Using GPS traces from a large fleet of taxis (approximately 16,000 vehicles), we extract a time-dependent, probabilistic model for each individual road segment. Our algorithms preprocess the road network equipped with probabilistic edge lengths into a data structure. Using this data structure, our query algorithm can efficiently answer approximate stochastic shortest-path queries. Our experiments validate the theoretical performance predictions, showing improvements upon existing query algorithms by several orders of magnitudes.

II. Background

A. Stochastic Shortest Paths

In this section, we provide a brief overview of the stochastic shortest-paths problem and known results, extracted from [18].

Consider a graph G with a given source node s and destination node t with stochastic edge delays w_e that come from independent distributions with means and variances (μ_e, τ_e), respectively. Denote the vector of edge delays by $\mathbf{w} = (w_1, ..., w_{|E|})$ and the corresponding vectors of edge means and variances by $\mu = (\mu_1, ..., \mu_{|E|})$ and $\tau = (\tau_1, ..., \tau_{|E|})$. Denote the incidence vector of a path by $\mathbf{x} \in \{0,1\}^{|E|}$ with ones corresponding to edges present in the path and zeros otherwise. Let the feasible set of s-to-t–paths be $\mathscr{F} \subset \{0,1\}^{|E|}$.

Given the uncertainty of edge delays, it is not immediate how to define a stochastic shortest path: is it the path that minimizes mean delay? Or should it minimize path variance or some other metric? Risk-averse users would naturally care about the variability in addition to mean delays. Two risk-averse models that have been previously considered are the *mean-risk model* and the *probability-tail model*, which we describe below.

The algorithms for both models make black-box calls to the underlying deterministic shortest-path algorithm, which computes

$$\text{min } \mathbf{w}^T\mathbf{x} \quad \text{subject to } \mathbf{x} \in \mathscr{F}. \tag{1}$$

a) A Mean-Risk Model: In this model, we seek the path minimizing mean delay plus a risk-aversion coefficient $c \geq 0$ times the path standard deviation, more formally:

$$\text{minimize } \mu^T\mathbf{x} + c\sqrt{\tau^T\mathbf{x}} \quad \text{subject to } \mathbf{x} \in \mathscr{F}. \tag{2}$$

This is a non-convex combinatorial problem, which can be solved exactly by enumerating all paths that minimize some positive combination of mean and variance (the latter is a deterministic shortest-path problem with respect to edge lengths equal to the corresponding mean–variance linear combination) and selecting the one with minimal objective (2) (see [18, 20] for more details). The number of deterministic shortest-path iterations is at most $n^{O(\log n)}$ in the worst case [20]. Furthermore, there is a fully-polynomial-time approximation scheme (FPTAS) for the problem, as stated in the following theorem:

Theorem 1 ([18, Theorem 4.1]): Suppose we have a δ–approximation algorithm for solving the deterministic shortest-path problem (1). For any $\varepsilon > 0$, the mean-risk model (2) can be approximated to a multiplicative factor of $\delta(1+\varepsilon)$ by calling the algorithm for the deterministic shortest-path problem polynomially many times in the input size and $1/\varepsilon$.

In other words, one can use both exact ($\delta = 1$) and δ–approximate (for any $\delta > 1$) shortest-path subroutines and obtain a corresponding approximate risk-averse solution.

b) Probability-Tail Model: In this model, we are given a deadline d and we seek the path maximizing the probability of arriving before the deadline: maximize $\Pr(\mathbf{w}^T\mathbf{x} \leq d)$ subject to $\mathbf{x} \in \mathscr{F}$. This model assumes normally-distributed edge delays in order to obtain a closed-form expression for the probability tail. In particular, when $\mathbf{w}$ is Gaussian, the problem is transformed into the following formulation [18]:

$$\text{maximize } \frac{d - \mu^T\mathbf{x}}{\sqrt{\tau^T\mathbf{x}}} \quad \text{subject to } \mathbf{x} \in \mathscr{F}. \tag{3}$$

Similarly to the mean-risk model, problem (3) can be solved exactly in time $n^{O(\log n)}$ [20] and it can be approximated as follows:

Theorem 2 ([18, Theorem 4.2]): Suppose we have a δ–approximation algorithm for solving the deterministic shortest-path problem (1). For any $\varepsilon > 0$, the probability tail model (3) has a $\sqrt{1 - \left[\frac{\delta - (1-\varepsilon^2/4)}{(2+\varepsilon)\varepsilon/4}\right]}$–approximation algorithm that calls the algorithm for the deterministic shortest-path problem polynomially many times in $1/\varepsilon$ and the input size, assuming the optimal solution to problem (3) satisfies $\mu^T x^* \leq (1-\varepsilon)d$ for a user-specified deadline d.

In our scenario, one main objective is to minimize the running time of the overall procedure at query time and hence the running times of these shortest-path subroutines are of high priority.

B. Approximate Distance Oracles

An (approximate) distance oracle is a data structure that efficiently answers shortest-path and/or distance queries in graphs. Relevant quantities of a distance oracle include the *preprocessing time*, which is the time required to construct the data structure, the *space* consumption, the *query time*, and, for approximate distance oracles, the *stretch*, which is the worst-case ratio among all pairs of query nodes of the query result divided by the actual shortest-path distance. Approximate distance oracles using quasi-linear space for poly-logarithmic query time and stretch $(1+\varepsilon)$ are known for planar [24] (implementation in [16]), bounded-genus [14], minor-free [1], geometric [13], and bounded-doubling-dimension graphs [4]. For realistic traffic scenarios, we need the distance oracle to work with directed graphs (also called *digraphs*). For planar digraphs, Thorup [24] gave an approximate distance oracle that

occupies space $O(n\varepsilon^{-1}\log(nL)\log(n))$ and answers $(1+\varepsilon)$–approximate distance queries in time $O(\varepsilon^{-1}+\log\log(nL))$, where L denotes the largest integer length (see [14] for different tradeoffs between space and query time).

At a high level, Thorup's method works as follows. The preprocessing algorithm recursively *separates* the graph into subgraphs of approximately equal size. *Shortest paths* are used as separators. For planar graphs, it is possible to separate the graph into two roughly balanced pieces by using three shortest paths Q_1, Q_2, Q_3. Next, the preprocessing algorithm considers each node $v \in V$ and it computes $O(1/\varepsilon)$ *portals* for v on each separator path Q_i such that the shortest path from v to each node $q \in Q_i$ is approximated within a $(1+\varepsilon)$–factor by going through one of the portals. Since each node only needs to remember $O(1/\varepsilon)$ distances to portals per level, and since each node participates in $O(\log n)$ levels, the overall space complexity per node is bounded by $O(\varepsilon^{-1}\log n)$. For directed planar graphs, the construction is more involved and the space and time complexities are slightly higher.

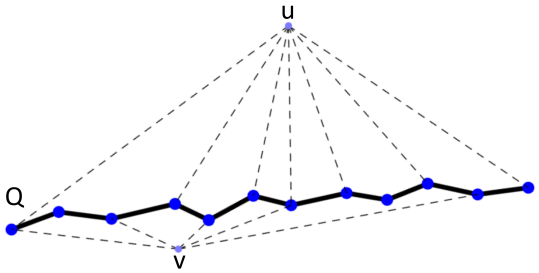

Fig. 2. For each node v we have $O(1/\varepsilon)$ *portals* on each separator path Q such that the shortest path from v to each node $q \in Q$ is approximated within a $(1+\varepsilon)$–factor by going through one of the portals. Note that two nodes u and v may not necessarily use the same set of portals.

Furthermore, many efficient practical methods have been devised [5, 10, 21], their time and space complexities are however difficult to analyze. Competitive worst-case bounds have been achieved under the assumption that actual road networks have small *highway dimension* [2, 3]. We point out one particular method, which is in some sense related to our construction. Geisberger, Kobitzsch, and Sanders [11] provide a heuristic shortest-path query method with edge lengths defined as a customizable linear combination of two edge length functions (this linear combination, however, is not arbitrary, which is why we cannot use their data structure).

III. Stochastic Motion Planning with Sparse Precomputed Data

A. Overview

We provide a new stochastic path planning algorithm and its implementation in a real traffic network. The main research objective is to minimize the time for querying a path. Algorithms studied in [15, 20] find an exact solution by iteratively determining the parameter value k that governs the edge length $\ell(e) := k \cdot \mu(e) + \tau(e)$ in terms of mean μ and variance τ. Such parameter values depend on a specific problem instance, defined by a deadline or a risk-aversion coefficient. We develop a method that finds a set of parameter values for the entire network, independent of the specific problem, and that provides guaranteed approximation accuracy for all choices of parameters *simultaneously*. We show that it suffices to maintain only a small set of parameters, which allows for small storage and efficient preprocessing. We preprocess the network using this set of parameter values.

We implement the deterministic shortest-path algorithms required in [18] using a distance oracle, as described in the previous section. This allows for fast query times. The preprocessing algorithm, while taking quasilinear time, is executed only once and the data structure can be computed on a powerful machine (not necessarily the machine where the query algorithm is run on). One important technical difference compared to the method described in [15] is that we cannot use an adaptive algorithm at query time, since the preprocessing algorithm must prepare for each step that might get executed at query time. In other words, the preprocessing algorithm must prepare for all possible query pairs $(s,t) \in V \times V$, and, potentially, deadline parameter values $d \in \mathbb{R}$. While such preprocessing algorithms are standard for classical graph distances, adaptations are necessary for stochastic shortest paths.

B. Preprocessing Algorithm

We adapt the algorithms in Theorem 1 and Theorem 2 to our scenario, where we have access to an approximate oracle for the deterministic shortest-path problem with edge lengths $e \mapsto k \cdot \mu(e) + \tau(e)$ for some $k \in \mathscr{K}$. In previous work [18], the aim was a polynomial-time algorithm and shortest-path problems could be computed in quasi linear time for *any* value k. Since we aim at sublinear query time, the queries made to the oracle need to be non-adaptive, or at least restricted to a small set of possible queries $\mathscr{K}$. We can then precompute distance oracles for all these values $k \in \mathscr{K}$. On a high level, the preprocessing algorithm is rather simple.

Algorithm 1: PREPROCESS

Data: $G = (V,E)$ with given lengths $\mu, \tau : E \to \mathbb{R}^+$
Result: distance oracles orc_k for each $k \in \mathscr{K}$
compute the set $\mathscr{K}$;
for $k \in \mathscr{K}$ **do**
  $orc_k \leftarrow$ build (approximate) distance oracle for length function $e \mapsto k \cdot \mu(e) + \tau(e)$;
end

Note that this preprocessing algorithm can be parallelized in a straightforward way. Since there are no dependencies between the oracles constructed for different values of k, the preprocessing algorithm can also make use of multi-core architectures.

As an immediate consequence, we obtain the following:

Claim 1: The preprocessing time is bounded by $O(|\mathscr{K}| \cdot \mathscr{P})$, where $\mathscr{P}$ denotes the preprocessing time of the deterministic distance oracle.

A main technical contribution of this work is showing how to compute (and analyze) this set $\mathscr{K}$. The size of $\mathscr{K}$ mostly depends on the user's choice of ε.

We run the deterministic preprocessing algorithm for the graph with edge lengths $\ell_k(e) = k \cdot \mu(e) + \tau(e)$ for $k \in \{L, (1+\xi)L, (1+\xi)^2 L, \ldots U\}$ for ξ, L, U to be defined for each model. Also, the approximation parameter ε we use for the preprocessing algorithm in the distance oracle changes with the model.

1) A Mean-Risk Model: Using a δ–approximate distance oracle, the end result is a $\delta(1+\varepsilon)$–approximation. Therefore, when asking for a $(1+\varepsilon)$–approximate answer, we internally set $\varepsilon_{int} := \varepsilon/3$, since, for $\varepsilon < 1/2$, we have that $(1+\varepsilon/3)^2 \leq 1 + 2\varepsilon/3 + \varepsilon^2/9 \leq 1+\varepsilon$. In the following, we write ε instead of ε_{int}. Next, we set ξ, L, and U as follows.

$$\begin{aligned} \xi &= \sqrt{\frac{\varepsilon}{1+\varepsilon}} \\ L &= \min_e \tau(e) \\ U &= \sum_e \tau(e) \leq n \cdot \max_e \tau(e) \end{aligned}$$

Alternatively, U can be set to the largest possible variance of any path. We have that the total number of values k is at most

$$|\mathscr{K}| = \log_{1+\xi}(U/L) = O\left(\log\left(n\frac{\max \tau(e)}{\min \tau(e)}\right)/\sqrt{\varepsilon}\right).$$

As a consequence, we obtain the following lemma.

Lemma 1: For any graph $G=(V,E)$ and for any two length functions $\mu : E \to \mathbb{R}^+$ and $\tau : E \to \mathbb{R}^+$, there exists a set $\mathscr{K}$ of size $|\mathscr{K}| = O\left(\varepsilon^{-1/2}\log\left(n\frac{\max \tau(e)}{\min \tau(e)}\right)\right)$ such that mean-risk shortest paths can be approximated by a shortest path in G with length function $\ell_k(e) = k \cdot \mu(e) + \tau(e)$ for some $k \in \mathscr{K}$.

2) Probability-Tail Model: For the probability-tail model, we follow the same strategy as in the previous section, based on the proof of [18, Lemma 3.4], showing that, in the mean–variance plane, any fixed parabola (also termed *level set*) with apex $(d,0)$ can be ε–approximated by a piecewise linear curve (see [18, Figure 1(b)] for an illustration) using few line segments. We exploit this ε–approximation in our algorithm. Furthermore, we define level sets *irrespective* of the deadline d: we may actually define $\mathscr{K}$ for any value of d — for a given query deadline d', the actual parabolas are just shifted to the left or to the right. Each segment defines a slope value k and the union of all these slope values (their absolute value, actually) is our set of slopes. Note that shifting the parabola to the left or to the right does not affect these slopes. We set ξ, L, and U as follows.

$$\begin{aligned} \xi &= \varepsilon/2 \\ L &= \frac{2\min_e \tau(e)}{d_L} \\ U &= \frac{2\sum_e \tau(e)}{\varepsilon d_U} \leq \frac{2n\max_e \tau(e)}{\varepsilon d_U} \leq \frac{2n\max_e \tau(e)}{\varepsilon \min_e \mu(e)} \end{aligned}$$

Although the user may choose to query the data structure for an arbitrarily large deadline d, we can give an upper bound on $d_L = \max d$ as follows. As soon as the deadline is so large that shortest paths do not change anymore, we do not have to consider larger values of d. In a straightforward way, d_L can be computed by computing all-pairs shortest paths for increasing values of d. Alternatively, again for increasing values of d, a 2–approximation of the diameter can be found in linear time.

C. Query Algorithm

The simplest version just tries all values $k \in \mathscr{K}$.

Algorithm 2: QUERY

Data: $G=(V,E)$ with given lengths $\mu, \tau : E \to \mathbb{R}^+$; distance oracles orc_k for each $k \in \mathscr{K}$; origin O, destination D, and mean risk coefficient c or deadline d depending on the risk model
Result: Approximately Optimal Path p

```
p ← NIL;
for k ∈ 𝒦 do
    q ← find_path(O, D, orc_k) ;
    if ( meanRisk and q.μ + c√(q.τ) < p.μ + c√(p.τ))
    or ( probTail and (d − q.μ)/(q.τ) > (d − p.μ)/(p.τ)) then
        p ← q;
    end
end
return p;
```

Claim 2: The query time is bounded by $O(|\mathscr{K}| \cdot \mathscr{Q})$, where $\mathscr{Q}$ denotes the query time of the deterministic distance oracle.

Combined with Claim 1, we obtain our main theorem.

Theorem 3: There is a data structure that can be computed in time $O(|\mathscr{K}| \cdot \mathscr{P})$, occupying space $O(|\mathscr{K}| \cdot \mathscr{S})$, and answering approximate stochastic shortest-path queries in time $O(|\mathscr{K}| \cdot \mathscr{Q})$, where $\mathscr{P}, \mathscr{S}, \mathscr{Q}$ denote the preprocessing, space, and query complexities of the deterministic distance oracle, respectively.

In the practical part, we also have a heuristic that binary searches for the optimal value in $\mathscr{K}$, resulting in faster query times.

Plugging in the distance oracle of Thorup [24], we obtain our main result. We state it in the mean-risk model. An analogous bound holds for the probability-tail model.

To apply [24, Theorem 3.16], we set the largest integer length $N^{(k)} := \max_{e \in E} k \cdot \mu(e) + \tau(e)$ for each preprocessing step $k \in \mathscr{K}$. Also, let $\bar{\tau} = \frac{\max \tau(e)}{\min \tau(e)}$.

Corollary 1: For any directed planar graph, there exists a data structure that can be computed in time $O(n(\log(n\bar{\tau}))(\lg(nN))(\lg n)^3\varepsilon^{-5/2})$, occupying space $O\left(n \cdot \varepsilon^{-3/2}(\lg n)(\lg(nN))(\log(n\bar{\tau}))\right)$, such that approximate stochastic shortest-path queries can be answered in time $O(\varepsilon^{-1/2}\log(n\bar{\tau})\lg\lg(nN) + \varepsilon^{-3/2}\log(n\bar{\tau}))$.

The space can be reduced at the cost of increasing the query time by using the construction in [14]. In particular, for scenarios where memory is scarce (e.g. on mobile devices),

one can obtain a *linear-space* data structure. Note that the preprocessing algorithm can be executed on a powerful computer and not necessarily on the mobile device. The query time remains poly-logarithmic.

IV. Stochastic Traffic Model based on GPS Traces

We compute a stochastic model of an entire city using the following process. We start with a heterogeneous set of approximately 500M anonymized GPS points. Grouped by individual vehicles, we extract those traces (subpaths), for which the sampling frequency is sufficiently high (at least one data point every thirty seconds). For each trace, we employ map-matching algorithms to compute the sequence of road segments that has the largest likelihood to have been used by this particular trace. The process of matching GPS traces data into a map has been investigated before [12, 27]. To overcome the noise and sparsity of GPS data, a map matching method based on the Viterbi algorithm was suggested in [17]. We use their method for our dataset. Since we do not have the groundtruth for our dataset, the correctness of the map-matching step cannot be verified. It is actually rather unlikely that all traces are mapped to the correct sequence of road segments. We assume that our dataset is large enough such that the majority of traces is mapped to correct road segments. We determine the speed of each vehicle per road segment and we sort all the values for each road segment by time. We then bin speed values into one-hour intervals for each day of the week. Using standard statistics, we compute the sample mean and standard deviation for the speed and travel time for each bin.

For examples of distributions for different road segments and different times of the day, see Fig. 3.

V. Practical Considerations

We adapt our method for planar networks so that it can efficiently handle the actual road network models we use in our experiments.

A. Non-planarities

Our network is not exactly planar and many real-world road networks may be non-planar [8]. The internal algorithms of the distance oracle use separator paths, which are sensitive to non-planarities (crossings). We propose to address most of these non-planarities as follows (inspired by similar constructions for minor-free graphs [1]). On the highest level of the recursion, instead of using two shortest paths to separate the graph, *(i)* we interpret the set of express ways (cf. highways) as a set of separator paths, *(ii)* we compute portals for each separator path Q and each node v, (distances to portals are computed with respect to the entire graph) and *(iii)* we remove the set of express ways from the graph. The highest level of the recursion handles all paths that use any part of an express way. The remaining graph has only few further crossings.

Turn restrictions: For some intersections, particularly in urban settings, it may not be possible to make a left turn due to a *turn restriction*, or it may just be more time-consuming to make a left turn, which can be modeled by a *turn cost*. Turn costs can be taken considered by modeling each road segment by a node of the graph, and then connecting only those nodes whose corresponding road segments can be traversed in sequence [7, 28]. Such a transformation increases the number of nodes [9] and, more importantly for our oracles, it also violates planarity. We argue that planarity is not just violated in a moderate way but that the violation is actually such that it cannot possibly be fixed. If arbitrary turn restrictions were allowed, we could just *(i)* draw any graph in the plane (for example the large-girth graphs in [25] or the expander graphs in [23]), *(ii)* add a node for each intersection, *(iii)* apply left-turn and right-turn restrictions (i.e., apply the supposedly-existing transformation without violating planarity), *(iv)* compute a planar distance oracle [24], and *(v)* store only the distance labels for the actual nodes of the graph. If the transformation in step *(iii)* could be done, the resulting oracle would contradict known lower bounds.

In our implementation, we connect each node to additional portals as follows: on any separator path, we deem each turn-restricted intersection as a portal (assuming that those intersections are not ubiquitous). The upper bound of $O(1/\varepsilon)$ portals per separator path cannot be guaranteed anymore. The bound in the implementation is $O(1/\varepsilon + R)$, where R is the number of intersections with turn restrictions on a given separator path.

B. Directions

Realistic modeling of traffic requires us to consider directed graphs.[1] The existing implementation [16] is for undirected graphs only and the directed oracle is significantly more involved [24]. Existing methods for directed road networks [5, 6, 10, 21] usually do not have theoretical worst-case bounds close to [24] (some theoretical results are known [2, 3]). For our scenario, we integrate the undirected implementation into our code. While our theoretical results hold for directed networks, our experimental evaluation is for undirected networks.

C. Reduced Set of Portals on Separator Paths

On the highest level, we cannot use planar-graph algorithms to efficiently compute portals and distances to portals for each node. Instead, we propose to relax the approximation guarantee in a controlled way. We heuristically treat each separator path (or express way) as in Algorithm 3.

D. Deadline values

In the probability-tail model, the user may specify an arbitrary deadline d. As a consequence, the ratio of the smallest and the largest possible d can be arbitrary large. As argued in

[1] The shortest-path problem for directed graphs is strictly more general. In a straightforward way, any undirected graph can be represented by a directed graph with bidirectional edges. For arbitrary directed graphs, even if they are guaranteed to be sparse, not much is known with respect to shortest-path queries, while the undirected variant is quite well understood.

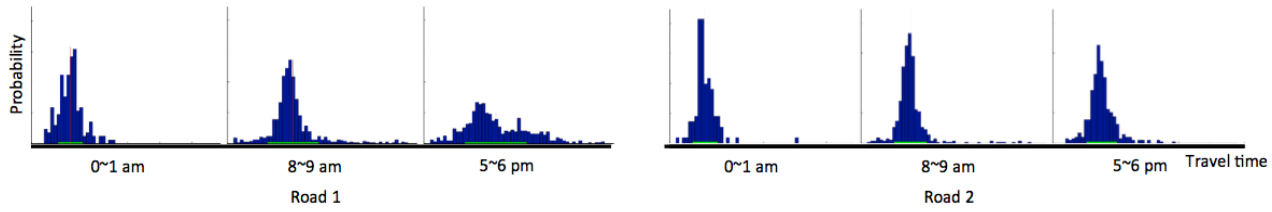

Fig. 3. The travel time distribution of two different road segments for $0 \sim 1$ am, $8 \sim 9$ am, and $5 \sim 6$ pm on weekdays.

Algorithm 3: PORTALS(H, Q, m)

Data: a path Q and a parameter m
Result: a set of portals for each $v \in V(H)$
SINGLESOURCESHORTESTPATHS(Q) ;
connect dummy source to each $q \in Q$;
computes first portal $q_0(v)$ and distance $d(v, q_0(v))$ for each $v \in V(H)$;
for $q_i \in Q$ **do**
  for $j \in \{0, 1, 2, \ldots m-1\}$ **do**
    $Q^{(j)} \leftarrow$ subset of nodes q_i where $i = j \mod m$;
    SINGLESOURCESHORTESTPATHS$(Q^{(j)})$;
    dummy source connected to all $q \in Q^{(j)}$;
    computes another portal $q_j(v)$ and distance $d(v, q_j(v))$ for each $v \in V(H)$;
    if $(1+\varepsilon)d(v, q_j(v)) > d(v, q_i(v)) + d(q_i(v), q_j(v))$ *for all portals* $q_i(v)$ *added previously* **then**
      add portal;
    end
  end
end

the theoretical sections, this can be controlled for effectively. In practical implementations, one could also limit the range of user input values by restricting the deadline values available.

VI. EXPERIMENTS

We evaluate an implementation of our algorithm on a real-world network dataset. We show that the query time is several orders of magnitude faster than the previous algorithms, and we also show that approximately optimal paths are reasonably close to optimal paths.

A. Setup

The tests were run on a machine with 2 GB of main memory and an Intel Xeon CPU 1.86GHz processor.

Our graph represents a city-scale network with 40,000+ nodes and 70,000+ edges. As described in the previous section, we adapted the worst-case-proven algorithm and tailored it to fit our real-world scenario. We implemented the distance oracle for undirected graphs based on the implementation of [16].

Since our network is mostly bidirectional (meaning that there are only few one-way streets), our graph could potentially be considered undirected. However, in time-dependent scenarios, the traffic may still flow in one direction, while a road may potentially be jammed in the other direction. Using the following heuristic, we can use the undirected variant of the distance oracle without significant impact on the path quality. To compile the different travel time distributions of bidirectional road segments, we built distance oracles for two different edge length functions: one with *inbound* driving time statistics and one with *outbound* driving time statistics. The geometric orientation of an edge with respect to the city center was used as the reference to indicate which of the two travel time statistics should be used when constructing the distance oracle. At query time, for example, if the route query overall corresponds to a *inbound* movement, the distance oracles built based on the *inbound* statistics are used.

We examined the pre-processing times, query times, and approximation ratios for different ε values for the two different models: Specifically, we used $\varepsilon \in \{0.1, 0.2, 0.3, 0.4, 0.5, 1\}$ for the Mean-Risk model, and $\varepsilon \in \{0.5, 1\}$ for the Probability-Tail model.

B. Pre-processing

Depending on the user's desired accuracy level ε, we find the values ξ, L, and U (as outlined in the theoretical part). The sizes of the set $\mathscr{K}$ for different values of ε are shown in Fig. 4 (Left). The pre-processing time depends mostly on $|\mathscr{K}|$ (and therefore on ε). For each value of ε, we used $c \in \{0, 0.5, 1, \cdots, 6\}$ for the Mean-Risk model, and $d \in \{1.1 \cdot m^*, 1.2 \cdot m^*, \cdots, 2 \cdot m^*\}$ for the Probability-Tail model, where m^* is the minimum expected travel time for a given origin-destination pair.

1) Computation time: The pre-processing times for different ε values are shown in Fig. 4 (Middle). For example, for the Mean-Risk model, if we set $\varepsilon = 0.1$, 82 different k values are used. The overall pre-processing time is 5,159 seconds. The average pre-processing time per oracle is about 63 seconds.

Note that the number of distance oracles to pre-process only depends on the desired accuracy of the solution paths defined by ε. It is independent of the users' other query parameters, such as origin, destination and the risk-aversion coefficient c in the Mean-Risk model or the deadline d in the Probability-Tail model.

2) Data space: Depending on ε, the overall space consumption varies greatly (see Fig. 4 (Right)), mostly due to two reasons. First, the number of k values increases as the accuracy is increased (meaning that ε is decreased), see Fig. 4 (Left), and second, the number of connections in the distance oracle grows linearly with $1/\varepsilon$.

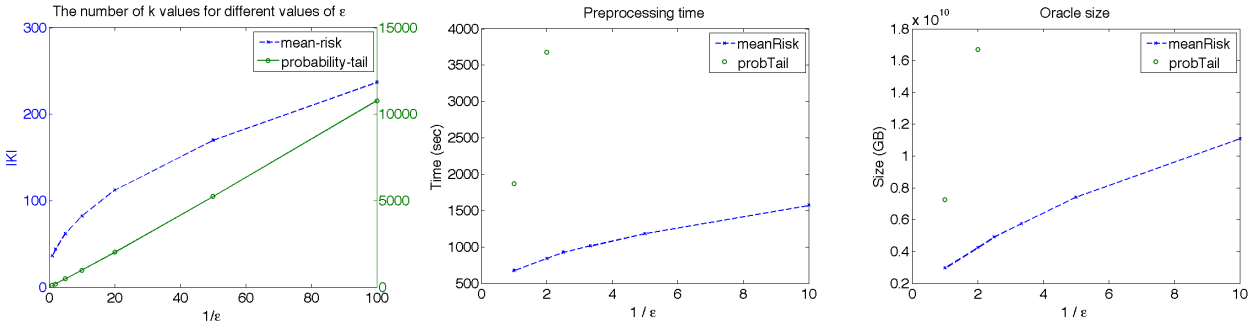

Fig. 4. The number of k values needed for the road network — this figure illustrates the growth of $|\mathscr{K}|$ for the mean-risk and the probability tail model with respect to the user-specified level of accuracy ε (Left), Pre-processing time for each ε (Middle), For each ε value ($1/\varepsilon$ on the x–axis), we report the total storage requirement for all the distance oracles (y–axis) (Right).

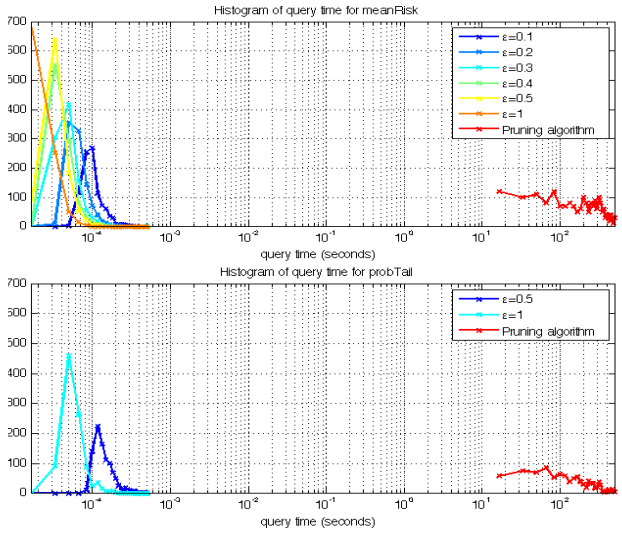

Fig. 5. The histogram of the query time for 1,000 random OD-paris. 'Pruning algorithm' refers to Algorithm 1 in [15]. Our new method is faster by several orders of magnitude. The speed-up is from tens of seconds or several minutes to tens of or hundreds of microseconds.

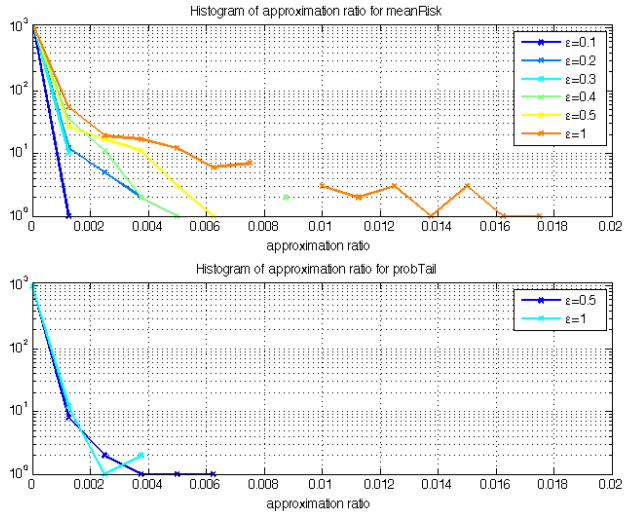

Fig. 6. The histogram of the absolute performance guarantee (error) for 1,000 random OD-paris. We used $c \in \{0, 0.5, \cdots, 6\}$ for the Mean-Risk model and $d \in \{1.1 \cdot m^*, 1.2 \cdot m^*, \cdots, 2 \cdot m^*\}$ for the Probability-Tail model, where m^* is the minimum expected travel time for a given origin-destination pair.

C. Path Queries

To measure the query time, we compute stochastic shortest paths between 1,000 random origin-destination (OD) pairs. To evaluate the quality and accuracy of the reported paths, we query 1,000 random OD pairs.

1) Query time: The time required to answer a route query for different values of ε is illustrated in Fig. 5. The query time decreases as ε increases. We implemented [15, Algorithm 1] as a comparison to our method in this paper. The query time results are plotted in Fig. 5 .

2) Approximation quality: We compare the quality of the path output by our algorithm to the optimal path length, computed by the algorithm described in [15]. As a quality measure, we list the approximation ratio, defined as the difference between the optimal solution (as defined in equations (2) and (3), respectively) and our solution, divided by the optimal solution (see Fig. 6). Our worst-case guarantee says that this ratio can never be larger than the user-specified level of accuracy ε. Our experimental results show that the ratio is smaller than ε, and, moreover, the ratio is substantially smaller than ε. Even for relatively large values such as $\varepsilon = 1$, we still obtain an approximation ratio less than 0.02. Based on this experimental result, we suggest using larger values of ε, saving both computation time (both pre-processing and query times) and storage (as outlined in the previous section).

D. Path Examples

We developed a web interface that responds to user inputs. Users can select the origin and destination, as well as the travel time of the day and the day of the week. The system computes the stochastic shortest path and displays it on the map. Fig. 1 shows our different solution paths for different deadlines for the Probability-Tail model with $\varepsilon = 0.1$.

VII. Conclusion

We improve upon the state of the art in stochastic path planning, providing a method that can answer stochastic shortest-path queries in poly-logarithmic time using a data structure that occupies space essentially proportional to the size of the

network. Our method has proven worst-case guarantees and it might be applicable to real-world scenarios.

VIII. Acknowledgments

Support for this research has been provided by the Singapore-MIT Alliance for Research and Technology (The Future of Urban Mobility project), NSF awards CPS-0931550 and 0735953, and ONR awards N00014-09-1-1051 (MURI SMARTS) and N00014-09-1-1031. C.S. was partially supported by the Swiss National Science Foundation. We are grateful for this support. Many thanks to Oliver Senn for connecting the first two authors.

References

[1] I. Abraham and C. Gavoille. Object location using path separators. In *25th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 188–197, 2006.

[2] I. Abraham, A. Fiat, A. Goldberg, and R. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

[3] I. Abraham, D. Delling, A. Fiat, A. Goldberg, and R. Werneck. VC-dimension and shortest path algorithms. In *38th Int. Colloquium on Automata, Languages, and Programming (ICALP)*, pages 690–699, 2011.

[4] Y. Bartal, L.-A. Gottlieb, T. Kopelowitz, M. Lewenstein, and L. Roditty. Fast, precise and dynamic distance queries. In *22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 840–853, 2011.

[5] H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast routing in road networks with transit nodes. *Science*, 316(5824):566, 2007.

[6] R. Bauer and D. Delling. SHARC: Fast and robust unidirectional routing. *ACM Journal of Experimental Algorithmics*, 14, 2009.

[7] T. Caldwell. On finding minimum routes in a network with turn penalties. *Comm. of the ACM*, 4(2), 1961.

[8] D. Eppstein and M. T. Goodrich. Studying (non-planar) road networks through an algorithmic lens. In *16th ACM SIGSPATIAL Int. Symposium on Advances in Geographic Information Systems (GIS)*, page 16, 2008.

[9] R. Geisberger and C. Vetter. Efficient routing in road networks with turn costs. In *10th Int. Symposium on Experimental Algorithms (SEA)*, pages 100–111, 2011.

[10] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *7th Int. Workshop on Experimental Algorithms (WEA)*, pages 319–333, 2008.

[11] R. Geisberger, M. Kobitzsch, and P. Sanders. Route planning with flexible objective functions. In *12th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 124–137, 2010.

[12] J. Greenfeld. Matching GPS observations to locations on a digital map. In *81st Annual Meeting of the Transportation Research Board*, volume 1, 2002.

[13] J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles for geometric spanners. *ACM Transactions on Algorithms*, 4(1), 2008.

[14] K. Kawarabayashi, P. N. Klein, and C. Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *38th Int. Colloquium on Automata, Languages and Programming (ICALP)*, pages 135–146, 2011.

[15] S. Lim, H. Balakrishnan, D. Gifford, S. Madden, and D. Rus. Stochastic motion planning and applications to traffic. *International Journal of Robotics Research*, 30 (6):699–712, 2011. Announced at WAFR 2008.

[16] L. F. Muller and M. Zachariasen. Fast and compact oracles for approximate distances in planar graphs. In *15th European Symposium on Algorithms (ESA)*, pages 657–668, 2007.

[17] P. Newson and J. Krumm. Hidden markov map matching through noise and sparseness. *17th ACM SIGSPATIAL Int. Conference on Advances in Geographic Information Systems (GIS)*, pages 336–343, 2009.

[18] E. Nikolova. Approximation algorithms for reliable stochastic combinatorial optimization. In *13th Int. Workshop on Approximation, Randomization, and Combinatorial Optimization (APPROX)*, pages 338–351, 2010.

[19] E. Nikolova, M. Brand, and D. R. Karger. Optimal route planning under uncertainty. In *16th Int. Conference on Automated Planning and Scheduling (ICAPS)*, pages 131–141, 2006.

[20] E. Nikolova, J. A. Kelner, M. Brand, and M. Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *14th European Symposium on Algorithms (ESA)*, pages 552–563, 2006.

[21] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest path queries. In *13th European Symposium on Algorithms (ESA)*, pages 568–579, 2005.

[22] R.-P. Schäfer. IQ routes and HD traffic: technology insights about TomTom's time-dynamic navigation concept. In *Foundations of Software Engineering (SIGSOFT FSE)*, pages 171–172, 2009.

[23] C. Sommer, E. Verbin, and W. Yu. Distance oracles for sparse graphs. In *50th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 703–712, 2009.

[24] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, 2004.

[25] M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005.

[26] TomTom. How TomTom's HD Traffic and IQ Routes data provides the very best routing — travel time measurements using GSM and GPS probe data. White Paper.

[27] C. White, D. Bernstein, and A. Kornhauser. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies*, 8(1-6):91–108, 2000.

[28] S. Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):363–380, 2002.