A New Formulation of The Shortest Path Problem with On-Time Arrival Reliability

Georgi Andonov  
Department of Computer Science  
Aalborg University, Denmark  
gandon16@student.aau.dk

Bin Yang  
Department of Computer Science  
Aalborg University, Denmark  
byang@cs.aau.dk

Abstract—We study stochastic routing in the PAth-CEntric (PACE) uncertain road network model. In the PACE model, uncertain travel times are associated with not only edges but also some paths. The uncertain travel times associated with paths are able to well capture the travel time dependency among different edges. This significantly improves the accuracy of travel time distribution estimations for arbitrary paths, which is a fundamental functionality in stochastic routing, compared to classic uncertain road network models where uncertain travel times are associated with only edges.

We investigate a new formulation of the shortest path with on-time arrival reliability (SPOTAR) problem under the PACE model. Given a source, a destination, and a travel time budget, the SPOTAR problem aims at finding a path that maximizes the on-time arrival probability. We develop a generic algorithm with different speed-up strategies to solve the SPOTAR problem under the PACE model. Empirical studies with substantial GPS trajectory data offer insight into the design properties of the proposed algorithm and confirm that the algorithm is effective.

Contributions: To the best of our knowledge, this is the first paper to study the SPOTAR problem in the PACE model. Second, we propose a generic algorithm with different speed-up heuristics to solve the problem. Third, we report on comprehensive experiments based on real world trajectory data.

I. INTRODUCTION

Emerging transportation innovations, such as mobility-on-demand services and autonomous driving, call for stochastic routing where travel time distributions, but not just average travel times, are utilized [1], [2]. For example, consider two paths $P_1$ and $P_2$ where both paths connect the same source and destination. The travel time distribution of the two paths are shown in Table I.

If only considering average travel times, $P_1$ is always the best choice, because $P_1$ has average travel time 49 whereas $P_2$‘s average travel time is 52. However, if a person needs to go to the destination within 60 minutes, e.g., to catch a flight, then $P_2$ is the best choice as it guarantees that the person will arrive the destination within 60 minutes. In contrast, taking $P_1$ may run into a risk that the person will miss the flight when the path takes 70 minutes.

In this paper, we consider stochastic routing that takes into account travel time distributions. In particular, we investigate the shortest path with on-time arrival reliability (SPOTAR) problem. Given a source, a destination, and a travel time budget, e.g., 60 minutes, the SPOTAR problem aims at finding a path that maximizes the on-time arrival probability, i.e., the probability of arriving at the destination within the time budget.

This report extends the paper “Stochastic Shortest Path Finding in Path-Centric Uncertain Road Networks”, to appear in IEEE MDM 2018, by providing a concrete running example of the studied SPOTAR problem in the PACE model and additional statistics of the used GPS trajectories in the experiments.

| Travel time (mins) | 40 | 50 | 60 | 70 |
|-------------------|----|----|----|----|
| $P_1$             | 0.5| 0.2| 0.2| 0.1|
| $P_2$             | 0  | 0.8| 0.2| 0  |

TABLE I: Travel Time Distributions for $P_1$ and $P_2$

II. RELATED WORK

We first review two uncertain road network models and then discuss existing studies on stochastic shortest path finding. 

Uncertain Road Network Modeling: The edge-centric uncertain road network model has been applied extensively in stochastic routing [4], [5], [7]–[10]. In the edge-centric model, not only edges, but some paths are also associated with uncertain travel times and the uncertain travel times that are associated with paths well capture the dependency of the travel times among different edges in the path. Thus, the PACE model is able to provide much more accurate travel time distributions [2], [3]. However, existing algorithms that work for the edge-centric model cannot be applied directly in the new PACE model. In this paper, we investigate how to solve the SPOTAR problem under the PACE model.

Contributions: To the best of our knowledge, this is the first paper to study the SPOTAR problem in the PACE model that exploits travel time dependencies among edges. First, we define the SPOTAR problem in the PACE model. Second, we propose a generic algorithm with different speed-up heuristics to solve the problem. Third, we report on comprehensive experiments based on real world trajectory data.
model, each edge is associated with an uncertain travel time and different edges’ uncertain travel times are independent. Given a path, the uncertain time of the path is derived by convoluting the uncertain times of the edges in the path. However, uncertain travel times among different edges in a path are often dependent but not independent. The edge-centric model is unable to capture the dependency and thus results in inaccurate uncertain travel times for paths. To contend with this, the Path-CENtric (PACE) model is proposed recently [2], [3]. PACE also associates some paths with joint travel time distributions that well capture the dependency of the uncertain travel times of the edges in the paths.

**Stochastic shortest path finding:** There exist two categories of stochastic shortest path finding—the Shortest path with on-time arrival reliability problem (SPOTAR) [4], [5]. SPOTAR identifies an a-priori path that maximizes the on time arrival reliability of reaching a destination within a pre-defined time budget. SOTA considers a scenario where a vehicle keeps moving and needs to choose which edge to follow at each intersection, based on the travel time has been already spent. SOTA tries to find an optimal policy that guides a vehicle to choose the next optimal edge at each intersection. We study the SPOTAR problem in the PACE model, while existing studies on SPOTAR all consider the edge-centric model.

### III. Preliminaries

We use a graph to model a road network. In particular, a road network is modeled as a directed graph $G = \langle V, E \rangle$. $V$ is a set of nodes that represent intersections and $E \subseteq V \times V$ is a set of edges that represent road segments. For example, Figure 1 shows a graph that models a road network with 6 intersections (i.e., nodes) and 9 road segments (i.e., edges).

A path is a sequence of adjacent edges $P = \langle e_1, e_2, \ldots, e_n \rangle$, where $n \geq 1$. In addition, we require that the edges in path $P$ are unique, meaning that $e_i \neq e_j$, if $1 \leq i, j \leq n$ and $i \neq j$. Next, we define a subsequence of edges in path $P$ as its sub-path $P'$. Formally, $P' = \langle e_i, e_{i+1}, \ldots, e_{j-1}, e_j \rangle$, where $1 \leq i < j \leq n$. For example, in Figure 1, path $P_1 = \langle e_1, e_4 \rangle$ is a sub-path of path $P_2 = \langle e_1, e_4, e_9 \rangle$.

Fig. 1: Road Network and Uncertain Edge Weights

Based on the above definitions, we proceed to introduce the classic edge-centric uncertain road network model and the path-centric uncertain road network model (i.e., PACE), respectively.

**Edge-centric uncertain road network model:** We maintain a weight function $W$ that takes as input an edge and returns an uncertain travel time for the edge. We maintain an uncertain travel time for each edge. The edge-centric model is considered as the de-facto model in stochastic routing [4]–[9], [16].

The uncertain edge weights are often instantiated using trajectories that occurred on the road network. Given a set of trajectories, we first break the trajectories to small pieces that fit the underlying edges and then use the small pieces to instantiate edge weights.

Assume that there we have 100 trajectories as shown in Table I. The first row of Table I means that 80 trajectories traversed path $\langle e_1, e_4 \rangle$, which spent 8 mins on $e_1$ and 6 mins on $e_4$.

| Traversed Path | Costs on edges | # of trajectories |
|----------------|----------------|------------------|
| $\langle e_1, e_4 \rangle$ | 8, 6           | 80               |
| $\langle e_1, e_4 \rangle$ | 10, 10         | 20               |
| $(e_1)$         | 8              | 100              |

**TABLE I:** Trajectory Examples

Now, we are able to instantiate $W(e_1) = \{(8, 0.9), (10, 0.1)\}$, meaning that $e_1$ may take 8 mins with probability 0.9, and 10 mins with probability 0.1, as shown in Fig. 1. This is because out of 200 trajectories that traversed $e_1$, 180 trajectories took 8 mins and 20 trajectories took 10 mins. Similarly, we have $W(e_4) = \{(6, 0.8), (10, 0.2)\}$.

In this edge-centric model, the uncertain travel times of edges are independent. Thus, the uncertain travel time of a path is computed as the convolution of the uncertain travel times of the edges in the path [4]–[9], [16]. For example, assuming that we have instantiated the edge weights as shown in Figure 1, consider path $P_1 = \langle e_1, e_4 \rangle$. Since we assume that the travel times of $e_1$ and $e_4$ are independent, we compute the joint travel time distribution first. Then, based on the joint distribution, we are able to compute the total travel time distribution, which is shown in Table II.

| $e_1$ | $e_4$ | Probability |
|-------|-------|-------------|
| 8     | 6     | 0.63        |
| 8     | 10    | 0.27        |
| 10    | 6     | 0.07        |
| 10    | 10    | 0.03        |

**TABLE II:** Total Travel Time Distribution

Note that the computed total travel time distribution of path $P_1$ is inconsistent with the “ground truth” distribution that is reflected in the trajectories in Table I. The ground truth distribution suggests $P_1$ took 14 mins with probability 0.8 (because out of 100 trajectories, 80 trajectories took 14 mins to traverse $P_1$) and 20 mins with probability 0.2 (because out of 100 trajectories, 20 trajectories took 20 mins to traverse $P_1$). This example shows that the travel time dependency, i.e.,...
traveling fast/slow on \( e_1 \) and also traveling fast/slow on \( e_2 \), is broken when using the edge-centric model, which results in inaccurate uncertain travel time for paths. It also shows that it is very important to use trajectories that are traversing the whole path, but not only intendant edges when calculating the travel cost distribution of a path.

Path-Centric (PACE) uncertain road network model: In the PACE model [2, 3], not only edges are associated with uncertain travel times, some paths are also associated with uncertain travel times. The uncertain travel times that are associated with paths model the joint travel time distributions of the edges in the paths.

Following the running example, PACE maintains not only uncertain travel times for edges, i.e., \( W(e_1) \) and \( W(e_4) \), but also the joint travel time distribution for path \( P_1 \), i.e., \( W(P_1) \), as shown Table IV(a). The joint travel time distributions maintained in PACE are directly derived from trajectories. Based on \( W(P_1) \), we are able to derive total travel time distribution of \( P_1 \), which well aligns with the ground truth distribution.

| \( e \) | \( W(e) \) | \( P \) | \( W(P) \) |
|---|---|---|---|
| 8 | 6 | 14 | 0.8 |
| 10 | 10 | 20 | 0.2 |

(a) \( W(P_1) \) (b) Total Travel Time Distribution

TABLE IV: Joint Distributions in PACE

Note that, in PACE, only some paths are associated with uncertain travel times, but not all paths. A road network may have a huge number of meaningful paths and thus we cannot afford maintain uncertain travel times for all paths in PACE. In addition, we often lack sufficient trajectories to cover all paths. In practice, we maintain joint distributions for those paths which have been traversed by sufficient amount of trajectories, e.g., popular paths.

Next, we present how to compute the travel time distribution of a path in PACE by using a concrete example. Assuming that, in addition to path travel time distribution \( W(P_1) \), PACE also maintains path travel time distribution \( W(P_2) \) for path \( P_2 = \langle e_4, e_9 \rangle \). Then, given a path \( P = \langle e_1, e_4, e_9 \rangle \), there exist more than one combination of weights such that each combination covers \( P \). For example, we may use \{\( W(e_1) \), \( W(e_4) \), \( W(e_9) \)\}, \{\( W(e_1) \), \( W(P_2) \)\}, \{\( W(P_1) \), \( W(e_9) \)\}, and \{\( W(P_1) \), \( W(P_2) \)\} to compute \( P \)'s uncertain travel time, respectively. It has been shown that the coarsest combination, i.e., the combination with the longest sub-paths, gives the most accurate uncertain travel time [2, 3]. In our example, it is \{\( W(P_1) \), \( W(P_2) \)\}.

After identifying the coarsest combination for a query path \( P \), say \{\( P_1, P_2, \ldots, P_n \)\}, we use Equation [1] to compute the joint distribution of \( P \) [2, 3].

\[
\text{prob}(P) = \frac{\prod_{i=1}^{n} W(P_i)}{\prod_{i=1}^{n-1} W(P_i \cap P_{i+1})}
\]

where \( P_i \cap P_{i+1} \) denotes the overlapped path of paths \( P_i \) and \( P_{i+1} \). Consider the running example where \( P = \langle e_1, e_4, e_9 \rangle \).

We have \( \text{prob}(P) = \frac{W(P_1) \cdot W(P_2)}{W(e_4)} \) since \( P_1 \cap P_2 = \langle e_4 \rangle \).

**Problem Definition:** Given a source vertex \( v_s \), a destination vertex \( v_d \), and a travel time budget \( T \), we aim at identifying path \( P^* \) which goes from \( v_s \) to \( v_d \) and has the largest probability of arriving the destination \( v_d \) within the time budget \( T \). This means that \( P^* = \arg \max_{P \in \text{Path}} \text{Prob}(P, \text{TravelTime} \leq T) \), where set \( \text{Path} \) contains all paths from \( v_s \) to \( v_d \) and the travel time distribution of path \( P \) is computed using the PACE model.

**IV. PROPOSED SOLUTION**

We propose an algorithm for solving the SPOTAR problem based on the PACE model. The algorithm is based on a heuristic function that estimates the least possible travel time from a vertex to the destination vertex. We first introduce the heuristic function and then introduce the algorithm using the heuristic function.

**A. Heuristic Function**

Motivated by \( A^* \) algorithm, to decide which edge we need to explore next to find the SPOTAR path, we maintain a heuristic function that takes as input a vertex and returns the least possible travel time from the vertex to the destination vertex \( v_d \).

A baseline heuristic function can simply return a travel time that equals to the Euclidean distance between the argument vertex and \( v_d \) divided by the maximum speed limit in the road network. However, this heuristic is too optimistic and thus loose. We aim at introducing a tighter heuristic.

To this end, we first introduce a reversed graph \( G_{\text{rev}} = (V, E') \) based on the original graph \( G = (V, E) \) and then compute a shortest path tree that is rooted at \( v_d \) in \( G_{\text{rev}} \). In particular, \( G_{\text{rev}} \) contains the same vertices with \( G \) and the edges in \( G_{\text{rev}} \) have reversed directions of the edges in \( G \). Fig. 2 shows the reversed graph of the graph shown in Fig. 1.

In the reversed graph, the edge weights are deterministic. In particular, for an edge \( e' = (v_i, v_j) \in E' \), its weight equals to the least travel time of the uncertain travel time on edge \( e = (v_j, v_i) \in E \) in the original graph. For example, edge \( e'_1 \) in \( G_{\text{rev}} \) has weight 8 since the least travel time of edge \( e_1 \) in \( G \) is 8.

![Fig. 2: Reversed Graph \( G_{\text{rev}} \) of \( G \)](image)
not need to compute a full shortest path tree but only part of the shortest path tree that contains the vertices whose distances to \( v_d \) are not larger than \( T \). We use function \( v_t \text{getMin}() \) to return the minimum travel time which is needed to travel from \( v_t \) to destination \( v_d \).

![Figure 3: The Reversed Shortest Path Tree With Root \( d \)](image)

Algorithm 1 SolveSPOTAR(Vertex \( s \), Vertex \( d \), TimeBudget \( T \))

1. Initialize a priority queue \( Q \).
2. Path \( P^* \leftarrow \text{null} \); double \( \text{maxProb} \leftarrow 0 \);
3. for \( v_t \in \text{all vertices that can be reached from } v_s \) do
4.  \( \text{siJoint} \leftarrow W(v_s, v_t) \); \( \text{siCost} \leftarrow W(v_s, v_t) \);
5.  if \( \text{MinCost} + v_t \text{getMin()} \leq T \) then
6.    \( r \leftarrow r_i(v_t, v_t) \rangle(T) \) based on Eq. 3.
7.    \( Q.\text{Push}(r, (v_s, v_t), \text{siJoint}, \text{siCost}) \);
8.  end if
9. end for
10. while \( Q \) is not empty do
11.    \( r, (P, \text{pathJoint}, \text{pathCost}) \) \( \leftarrow Q.\text{ExtractMax}() \);
12.    \( v_i \leftarrow \text{last vertex in path } P \);
13.    if \( v_i = v_d \) then
14.        if \( r > \text{maxProb} \) then
15.            \( P^* \leftarrow P \); \( \text{maxProb} \leftarrow r \);
16.        Delete all elements whose \( r \) values are smaller than the current \( r \) from \( Q \);
17.        Continue;
18.    else
19.        Break;
20.    end if
21.    end if
22.    for \( v_k \in \text{all vertices that can be reached from } v_i \) do
23.        if Path \( P \) has not traversed \( v_k \) yet then
24.            \( \text{extP} \leftarrow P \langle (v_i, v_k), v_k \rangle \);
25.            \( \text{ikMin} \leftarrow \text{MinCost}(v_i, v_k) \);
26.            \( \text{pathMin} \leftarrow \text{MinCost}(\text{pathCost}) \);
27.            if \( \text{ikMin} + \text{pathMin} + j \text{getMin()} \leq T \) then
28.                \( \text{extPJoint} \leftarrow \text{PACE}(\text{extP}) \);
29.                \( \text{extPCost} \leftarrow \text{JointToCost}(\text{extPJoint}) \);
30.                \( r \leftarrow r_k(T) \) based on Eq. 3.
31.                if CheckStochasticDominance() then
32.                    \( Q.\text{Push}(r, (\text{extP}, \text{extPJoint}, \text{extPCost})) \)
33.                end if
34.            end if
35.        end if
36.    end for
37. end while
38. return path \( P^* \);

Given a path \( P \) from source vertex \( v_s \) to vertex \( v_i \), we are able to compute the cost distribution of path \( P \) using the PACE model. Now, based on best travel time distribution \( u_t(t) \), we are able to estimate the best travel time distribution if we follow \( P \) and continue to reach the destination \( v_d \). In particular, we use \( r_i^P(x) \) to denote the largest probability of arriving the destination \( v_d \) within \( x \) mins if we follow path \( P \) to go from \( v_s \) to \( v_i \) and then reach the destination \( v_d \).

\[
    r_i^P(x) = \sum_{k=1}^{x} probP(k).u_t(x-k) \tag{3}
\]

where \( probP(k) \) indicates the probability that \( P \) takes \( k \) mins and \( u_t(x-k) \) indicates the probability that it takes less than \( x-k \) mins to go from \( v_t \) to \( v_d \).

B. Proposed Algorithm

The algorithm for computing the SPOTAR problem is shown in Algorithm 1.

We start initializing a priority queue \( Q \) that holds elements in the form of \( (r, (P, \text{pathJoint}, \text{pathCost})) \), where \( P \) is a path from source \( v_s \) to some vertex \( v_i \), \( r = r_i^P \langle(T) \) is the largest probability of arriving \( v_d \) within time budget \( T \) using path \( P \), \( \text{pathJoint} \) is the joint distribution of \( P \), and \( \text{pathCost} \) is the total travel time distribution of \( P \). We also initialize path \( P^* \), which intends to maintain a candidate SPOTAR path and its probability \( \text{maxProb} \) of arriving destination \( v_d \).

We first check all vertices that are from vertex \( v_s \) (line 8) to line 11. Since now the candidate path \( (v_s, v_i) \) only has one edge, the joint distribution and cost distribution are the same (line 4). Next, if the minimum travel time from the candidate path to \( v_d \) is already larger than the time budget \( T \), we do not insert it into the priority queue. Otherwise, we insert it.

We proceed to the main while loop at line 10 in each iteration, we extract the element with the largest \( r \) value until the priority queue \( Q \) is empty.

Assume the element with the largest \( r \)-value represents path \( P \). We check if \( P \) has already arrived destination \( v_d \). If yes, we check if \( P \)'s \( r \)-value is larger than the current maximum probability \( \text{maxProb} \). If yes, we update the candidate SPOTAR path \( P^* \) and remove all the elements from \( Q \) if their \( r \)-values are smaller than the current \( r \). If not, we can stop the algorithm since all the remaining paths in the priority queue have \( r \)-values less than the current maximum probability.
max Prob. This means it makes no sense to keep exploring the remaining paths, since their probabilities of arriving \( v_d \) within time budget \( T \) cannot be greater than \( \text{max} \text{Prob} \).

On the other hand, if \( P \) has not arrived at \( v_d \) yet, i.e., \( v_i \neq v_d \), we need to extend \( P \) by one more edge \((v_i, v_k)\) to get an extended path \( \text{ext}P \), which arrives vertex \( v_k \). Here, we only consider simple path without cycles as it has been shown that a path with cycles cannot be a SPOTAR path [6].

If it is possible to follow \( \text{ext}P \) to arrive \( v_d \) within the time budget \( T \), we compute its joint distribution \( \text{ext}P \text{Joint} \) using the PACE model and its total cost distribution \( \text{ext}P \text{Cost} \). We next apply Equation 3 to compute the \( r \)-value of the extend path \( \text{ext}P \).

Before inserting the extended path \( \text{ext}P \) into the priority queue \( Q \), we need to conduct the following stochastic dominance check. If there already exists a path \( P_k \) in \( Q \) which also arrives \( v_k \). We check if \( \text{ext}P \)’s cost distribution stochastic dominates \( P_k \)’s cost distribution. If yes, we remove \( P_k \)’s element from \( Q \) and insert \( \text{ext}P \) into \( Q \). Similarly, if \( P_k \) stochastic dominates \( \text{ext}P \), we do not need to insert \( \text{ext}P \) into \( Q \). Note that only considering if \( \text{ext}P \)’s \( r \)-value is larger than \( P_k \)’s \( r \)-value is insufficient [4], [8].

Finally, we return the SPOTAR path \( P^* \) along with the maximum probability of arriving \( v_d \) within time budget \( T \).

V. Running example

This section presents a concrete example of the SPOTAR problem which is solved by the proposed algorithm. The graph from Figure 1 is used to show a running example of the Algorithm for computing SPOTAR under the PACE model [1]. The joint distribution and the derived cost distribution for paths \( < e_1, e_4 > \) and \( < e_2, e_6 > \) for the graph in Fig. 1 are shown in Table V and VI respectively. For example, the joint distribution for path \( < e_1, e_4 > \) has been derived from the trajectories shown in Table II.

\[
\begin{array}{|c|c|c|}
\hline
\text{edge} & \text{probability} \\
\hline
e_1 & 8 & 0.8 \\
\hline
e_4 & 10 & 0.2 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{path} & \text{probability} \\
\hline
P_{e_1, e_4} & 14 & 0.8 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{joint dist} & \text{total dist} \\
\hline
\text{Table V: Joint vs. total cost distributions for path } < e_1, e_4 > \\
\hline
\text{edge} & \text{probability} \\
\hline
e_2 & 8 & 0.5 \\
\hline
e_6 & 11 & 0.3 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{path} & \text{probability} \\
\hline
P_{e_2, e_6} & 13 & 0.7 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{Table VI: Joint vs. total cost distributions for path } < e_2, e_6 > \\
\hline
\text{joint dist} & \text{total dist} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{a) } W(P_{<e_1,e_4>}) & \text{b) Total Cost Distribution} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{a) } W(P_{<e_2,e_6>}) & \text{b) Total Cost Distribution} \\
\hline
\end{array}
\]

**Problem:** Consider a problem that aims at finding the path with the highest probability of reaching destination \( d \) from source \( s \) in a graph \( G \) within a time budget \( T = 22 \). In addition, we have \( \delta = 1 \); and the following paths’ joint distributions are known a priori and is maintained in the path weight function \( W: < e_1, e_4 > \) in Table V; \( < e_2, e_6 > \) in Table VI.

**Solution:** The reversed graph with minimum travel times for all edges is shown in Figure 2. Next, the algorithm computes the shortest path tree based on the graph, and labels each node with the distance to destination node \( d \) as shown in Figure 3. For each node \( i \), we define \( u_i(t) \) as the highest probability of reaching the destination node \( d \) from node \( i \), i.e., Equation 4 as stated in [1], where \( i.\text{GetMinVal}() \) denotes the minimum travel time from node \( i \) to destination node \( d \) and \( t \) is a given time budget.

\[
u_i(t) = \begin{cases} 1 & \text{if } t \geq v_i.\text{GetMinVal()} \\ 0 & \text{if } t < v_i.\text{GetMinVal()} \end{cases}
\]

(4)

The algorithm begins with initialization of the priority queue \( Q \). Next, it iterates over all adjacent arcs of \( s \), i.e. \( e_1 \) and \( e_2 \). For arc \( e_1 \) the algorithm computes \( \text{prob}_{e_1} \), which is given as follows:

\[\text{prob}_{e_1} = \{(8, 0.9), (10, 0.1)\}\]

Next, the value of \( r \) is computed using \( u_r \) as follows:

\[r = \sum_{k=1}^{T} \text{prob}_{e_1}(k).u_r(T - k) = \sum_{k=1}^{22} \text{prob}_{e_1}(k).u_r(22 - k) = \text{prob}_{e_1}(8).u_r(22 - 8) + \text{prob}_{e_1}(10).u_r(22 - 10) = 0.9 * 1 + 0.1 * 1 = 1\]

After the value of \( r \) has been computed, edge \( e_1 \) has been added to the path created until now, i.e., \( < e_1 > \). The path along with its \( r \)-value are pushed to the priority queue \( Q \).

\[Q.\text{push}(1.0, (< e_1 >, \text{prob}_{e_1}, \text{cost}_{e_1}))\]

The second adjacent arc of node \( s \) that has been considered by the algorithm is arc \( e_2 \). The algorithm again starts by computing the joint probability for the path discovered until now (i.e. \( e_2 \)), which is shown as follows.

\[\text{prob}_{e_2} = \{(8, 0.2), (11, 0.8)\}\]

Next, the algorithm computes \( r \)-values for node \( r \) using \( u_r = 10 \).

\[r = \sum_{k=1}^{T} \text{prob}_{e_2}(k).u_r(T - k) = \sum_{k=1}^{22} \text{prob}_{e_2}(k).u_r(22 - k) = \text{prob}_{e_2}(8).u_r(22 - 8) + \text{prob}_{e_2}(11).u_r(22 - 11) = 0.2 * 1 + 0.8 * 1 = 1\]

Once the \( r \)-value of \( r \) has been computed, edge \( e_2 \) has been added to the path created until now and its value has been pushed to the priority queue \( Q \)

\[Q.\text{push}(1.0, (< e_2 >, \text{prob}_{e_2}, \text{cost}_{e_2}))\]

Next, the algorithm has to pop the value with the maximum key from the priority queue \( Q \). Until now there are two elements in \( Q \) i.e. edge \( e_1 \) and edge \( e_2 \) both with key value equal to 1. The algorithm decides to pop the element which is associated with the path \( e_1 \) based on the key (1.0).
After that, the algorithm iterates over all adjacent arcs of node \( e \) i.e. arcs \( e_5 \) and \( e_4 \).

For arc \( e_5 \) conditional independence holds because there are no overlapping sub-paths that contains \( e_5 \) and the path constructed so far by the algorithm. Therefore, the algorithm computes the joint distribution and then derive the associated cost distribution as shown in Table VII:

\[
\text{Table VII: Joint and the derived cost distributions for path } \left< e_1, e_5 \right>
\]

| \( e_1 \) | \( e_5 \) | Probability |
|-----|-----|--------|
| 8   | 8   | 0.72   |
| 8   | 10  | 0.18   |
| 10  | 8   | 0.08   |
| 10  | 10  | 0.02   |

\[
\text{Table VIII: Joint vs. total cost distributions for path } \left< e_1, e_4, e_9 \right>
\]

| \( e_1 \) | \( e_4 \) | \( e_9 \) | Probability |
|-----|-----|-----|--------|
| 8   | 6   | 5   | 0.32   |
| 8   | 6   | 9   | 0.48   |
| 10  | 10  | 5   | 0.08   |
| 10  | 10  | 9   | 0.12   |

\[
\text{Fig. 4: Stochastic dominance between paths } \left< e_1, e_2 \right> \text{ and } \left< e_2, e_6 \right>. \text{ None of the paths stochastically dominates the other one.}
\]

As it can be seen from Figure 4, none of the path stochastically dominates the other, therefore we cannot prune either of them, hence the algorithm adds path \( \left< e_2, e_6 \right> \) to the priority queue \( Q \).

\[
\text{Q.push(0.70, } \left< e_2, e_6 \right>, \text{prob}_{e_2,e_6}, \text{cost}_{e_2,e_6})\)
\]

At this point the priority queue \( Q \) contains two paths: \( \left< e_1, e_4 \right> \) and \( \left< e_2, e_6 \right> \) with the following objects associated as follows:

\[
(0.80, \left< e_1, e_4 \right>, \text{prob}_{e_1,e_4}, \text{cost}_{e_1,e_4})
\]

Path \( \left< e_1, e_4 \right> \) has been popped up from the priority queue \( Q \) based on its key 0.8 and for all adjacent edges of node \( q \) i.e. arcs \( e_7 \) and \( e_9 \).

Arc \( e_7 \) is pruned by the algorithm bases on its minimum travel time to destination calculated as follows: \( (14+13)+8 > T = 22 \).

Arc \( e_9 \) is explored by the algorithm, it computes the travel cost distribution for path \( \left< e_1, e_4, e_9 \right> \) with indepedence between \( \left< e_1, e_4 \right> \) and \( \left< e_9 \right> \) does hold, the result can be seen in Table VIII:

\[
\text{Table VIII: Joint vs. total cost distributions for path } \left< e_1, e_4, e_9 \right>
\]

| \( e_1 \) | \( e_4 \) | \( e_9 \) | Probability |
|-----|-----|-----|--------|
| 8   | 6   | 5   | 0.32   |
| 8   | 6   | 9   | 0.48   |
| 10  | 10  | 5   | 0.08   |
| 10  | 10  | 9   | 0.12   |

The search reaches a destination node \( d \). Solution 1 for path \( \left< e_1, e_4, e_9 \right> \) with probability 0.32 of reaching the destination \( d \) within a time budget of 22 units. After obtaining a solution. The algorithm checks the keys (i.e., the \( r \)-values) of all elements in priority queue. If the key is less then the probability of the solution (0.32), we prune the corresponding entry from the priority queue. Since 0.70 > 0.32, we therefore
cannot prune path \(<e_2, e_6>\) which is in the priority queue.

Pop \(<e_2, e_6>\) based on key 0.70. For all adjacent arcs of node 2, First \(e_7\) can be pruned based on minimum travel time \((13 + 13) + 8 > T = 22\) Second, arc \(e_9\). The algorithm computes the travel cost for the path \(<e_2, e_6, e_9>\).

Independence does hold, therefore we have that

\[
prob_{e_2, e_6, e_9} = prob_{e_2, e_6} \cdot prob_{e_9}
\]

The result is shown in Table IX.

| \(e_2\) | \(e_6\) | \(e_9\) | Probability |
|---|---|---|---|
| 8  | 5  | 5  | 0.28 |
| 8  | 5  | 9  | 0.42 |
| 11 | 9  | 5  | 0.12 |
| 11 | 9  | 9  | 0.18 |

| \(P_{e_2, e_6, e_9}\) | Probability |
|---|---|
| 18 | 0.28 |
| 22 | 0.42 |
| 25 | 0.12 |
| 29 | 0.18 |

TABLE IX: Joint vs. total cost distributions for path \(<e_2, e_6, e_9>\)

The search reaches destination node \(d\). Solution 2 for path \(<e_2, e_6, e_9>\) with probability 0.70 of reaching the destination node \(d\) within a time budget 22 units. We check the keys of all elements in priority queue. If the key is less than 0.70, we eliminate the corresponding entry from the priority queue. Priority queue is empty, so we cannot prune. Since solution 2 with path \(<e_2, e_6, e_9>\) has higher probability of reaching the destination node \(d\) within a time budget of 22 units. It is the final solution.

VI. EXPERIMENTAL RESULTS

We report on experimental results using real world GPS trajectories.

A. Experimental Setup

We use the road network of Aalborg, Denmark, which contains 6,253 nodes and 10,716 edges. We use 37 million GPS records that occurred in Aalborg from Jan 2007 to Dec 2008 to instantiate the PACE model. The sampling rate of GPS records is 1 Hz (i.e., one GPS record per second). If an edge is not covered by GPS data, we use the length of the edge and the speed limit on the edge to derive a travel time. If a path has been traversed by 10 trajectories, we maintain a joint distribution for the path. A visual representation of the edges that are covered and not covered by the GPS trajectories can be found in Figure 5.

Queries: We consider different settings to generate SPOTAR queries. We vary time budget (seconds) from 300, 500, 700, to 1,000. We also vary the Euclidean distance (km) between source-destination pairs: [0, 1], [1, 2], [2, 3], and [3, 4]. For each setting, we randomly generate 20 source-destination pairs.

Methods: We consider different heuristic functions: (1) the proposed solution with minimum travel time to the destination using shortest path trees (SP); (2) the baseline heuristic using Euclidean distance divided by the maximum speed limit (BA).

We apply the both heuristics on both PACE and the edge-centric model (EDGE), resulting four methods: SP+PACE, SP+EDGE, BA+PACE, and BA+EDGE.

Evaluation Metrics: We report average runtimes and sizes of search space for running SPOTAR in different settings.

We conduct experiments on a computer with Intel® Core™ i5-4210U CPU @ 1.70GHz × 4 processors with 12GB RAM under 64 bit Linux Fedora 25. The code was implemented in Python 3.

B. Experimental Results

Runtime: Figure 6 shows the runtimes when using four methods under different settings. When the distance between a source-destination pair increases, the runtimes of all methods also increases for all time budgets. The SP heuristic shows significantly better runtime in comparison with the BA heuristic under all settings on both models. In addition, it is clear than the runtime growth of the BA heuristic is much faster in comparison with the SP heuristic as the distance between a source and a destination increases. Under the same heuristic function and time budget, methods based on PACE have similar runtime with the methods based on EDGE. This suggests that although PACE is able to provide more accurate results that EDGE, it does not takes longer run time. This suggests that PACE is both accurate and efficient.

Search space: We investigate the sizes of search spaces that different methods explore. In particular, we define the search space as the edges that have been explored by a method.

We compare the search spaces that are explored by different methods. Figure 7 shows that the search space of BA is much higher than that of SP in all settings, which is consistent with the performance of runtime shown in Figure 6.

Next, we show visually the edges that are discovered by both heuristic under a specific source-destination pair setting...
VII. Conclusions and Outlook

Arriving on time is an important problem that has many applications in intelligent transportation systems. We present an effective algorithm that solves the SPOTAR problem on the novel path-centric PACE model that considers the dependencies among travel times on different edges. Experimental results on real-world trajectories suggest that the proposed algorithm is effective. In the future, we plan to study further speed-up strategies, e.g., using contraction hierarchies and hub labeling, on top of the PACE model, also possibly using parallel computing [17].
REFERENCES

[1] C. Guo, C. S. Jensen, and B. Yang. “Towards Total Traffic Awareness.” SIGMOD Record, vol. 43, no. 3, pp. 18–23, 2014.

[2] J. Dai, B. Yang, C. Guo, C. S. Jensen, and J. Hu. “Path Cost Distribution Estimation Using Trajectory Data.” PVLDB 10(3): 85-96, 2016.

[3] B. Yang, J. Dai, C. Guo, C. S. Jensen, and J. Hu. “PACE: a PAth-CEntric paradigm for stochastic path finding.” VLDB Journal 27(2): 153-178, 2018.

[4] M. Niknami, S. Samaranayake. “Tractable Pathfinding for the Stochastic On-Time Arrival Problem.” SEA, 231-245, 2016.

[5] G. Sabran, S. Samaranayake, A. Bayen. “Precomputation techniques for the stochastic on-time arrival problem.” ALENEX, 138-146, 2014.

[6] Yu Nie and Xing Wu. “Shortest path problem considering on-time arrival probability.” Transportation Research Part B: Methodological, 2009.

[7] C. Guo, B. Yang, O. Andersen, et al. “EcoSky: Reducing vehicular environmental impact through eco-routing.” ICDE, 1412-1415, 2015.

[8] B. Yang, C. Guo, C. S. Jensen, et al. “Stochastic skyline route planning under time-varying uncertainty.” ICDE, 136–147, 2014.

[9] B. Yang, M. Kaul, and C. S. Jensen. “Using incomplete information for complete weight annotation of road networks.” TKDE, 26(5):1267–1279, 2014.

[10] C. Guo, B. Yang, J. Hu, C. S. Jensen. “Learning to Route with Sparse Trajectory Sets.” ICDE, 12 pages, 2018.

[11] H. Liu, C. Jin, B. Yang, A. Zhou. “Finding Top-k Optimal Sequenced Routes.” ICDE, 12 pages, 2018.

[12] H. Liu, C. Jin, B. Yang, A. Zhou. “Finding Top-k Shortest Paths with Diversity.” TKDE, 30(3):488-502, 2018.

[13] J. Hu, B. Yang, C. Guo, C. S. Jensen. “Risk-aware path selection with time-varying, uncertain travel costs: a time series approach.” The VLDB Journal 27(2):179-200, 2018.

[14] J. Hu, B. Yang, C. S. Jensen, Y. Ma. “Enabling time-dependent uncertain eco-weights for road networks.” GeoInformatica 21(1):57-88, 2017.

[15] Z. Ding, B. Yang, R. H. Güting, Y. Li. “Network-Matched Trajectory-Based Moving-Object Database: Models and Applications.” IEEE Trans. Intelligent Transportation Systems 16(4):1918-1928, 2015.

[16] B. Yang, C. Guo, Y. Ma, and C. S. Jensen. “Toward personalized, context-aware routing.” The VLDB Journal, 24(2):297–318, 2015.

[17] B. Yang, Q. Ma, W. Qian, A. Zhou. TRUSTER: “TRajectory Data Processing on CIUSTERS.” DASFAA, 768-771, 2009.

[18] Z. Ding, B. Yang, Y. Chi, L. Guo. “Enabling Smart Transportation Systems: A Parallel Spatio-Temporal Database Approach.” IEEE Trans. Computers 65(5): 1377-1391, 2016.