



# Efficient Stochastic Routing in Path-Centric Uncertain Road Networks

Chenjuan Guo  
East China Normal  
University  
cjguo@dase.ecnu.edu.cn

Ronghui Xu  
East China Normal  
University  
rhxu@stu.ecnu.edu.cn

Bin Yang  
East China Normal  
University ✉  
byang@dase.ecnu.edu.cn

Ye Yuan  
Aalborg University  
yuanye@cs.aau.dk

Tung Kieu  
Aalborg University  
tungkvt@cs.aau.dk

Yan Zhao  
Aalborg University  
yanz@cs.aau.dk

Christian S. Jensen  
Aalborg University  
csj@cs.aau.dk

## ABSTRACT

The availability of massive vehicle trajectory data enables the modeling of road-network constrained movement as travel-cost distributions rather than just single-valued costs, thereby capturing the inherent uncertainty of movement and enabling improved routing quality. Thus, stochastic routing has been studied extensively in the edge-centric model, where such costs are assigned to the edges in a graph representation of a road network. However, as this model still disregards important information in trajectories and fails to capture dependencies among cost distributions, a path-centric model, where costs are assigned to paths, has been proposed that captures dependencies better and provides an improved foundation for routing. Unfortunately, when applied in this model, existing routing algorithms are inefficient due to two shortcomings that we eliminate. First, when exploring candidate paths, existing algorithms only consider the costs of candidate paths from the source to intermediate vertices, while disregarding the costs of travel from the intermediate vertices to the destination, causing many non-competitive paths to be explored. We propose two heuristics for estimating the cost from an intermediate vertex to the destination, thus improving routing efficiency. Second, the edge-centric model relies on stochastic dominance-based pruning to improve efficiency. This pruning assumes that costs are independent and is therefore inapplicable in the path-centric model that takes dependencies into account. We introduce a notion of virtual path that effectively enables stochastic dominance-based pruning in the path-based model, thus further improving efficiency. Empirical studies using two real-world trajectory sets offer insight into the properties of the proposed solution, indicating that it enables efficient stochastic routing in the path-centric model.

## PVLDB Reference Format:

Chenjuan Guo, Ronghui Xu, Bin Yang, Ye Yuan, Tung Kieu, Yan Zhao, and Christian S. Jensen. Efficient Stochastic Routing in Path-Centric Uncertain Road Networks. PVLDB, 17(11): 2893 - 2905, 2024.  
doi:10.14778/3681954.3681971

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing [info@vldb.org](mailto:info@vldb.org). Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.  
Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.  
doi:10.14778/3681954.3681971

## PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/decisionintelligence/Route-sota>.

## 1 INTRODUCTION

Emerging innovations in transportation with disruptive potential, such as autonomous vehicles, transportation-as-a-service, and coordinated fleet transportation, call for high-resolution routing. For example, PostNord<sup>1</sup>, a logistics service provider, and FlexDanmark<sup>2</sup>, a fleet transportation organization, often make deliveries with associated travel cost budgets, e.g., 2 hours. High-resolution routing enables them to maximize arrivals within given time budgets when scheduling trips, thereby improving the quality of their services [7].

Meanwhile, the availability of increasingly massive volumes of vehicle trajectory data enables opportunities for capturing travel costs, such as travel time, at an unprecedented level of detail. The state-of-the-art models capture travel costs as distributions. This level of detail enables the above services that are not possible when using only average values. As another example, Table 1 shows the travel-time distributions of two paths  $P_A$  and  $P_B$  from an office to an airport, scheduled for an autonomous vehicle. The average travel times, i.e., weighted sum, of  $P_A$  and  $P_B$  are 49 and 52, respectively. If a person needs to arrive at the airport in 60 minutes, taking  $P_A$  is more risky since it incurs a 10% chance of being late, although it has a smaller average. Given a source and a destination on an uncertain road network, a departure time, and a travel cost budget, e.g., 60 minutes, we study the classical arriving on time problem [2, 22], which conducts stochastic routing and aims to find a path that maximizes the probability of arriving at the destination within the cost budget.

Table 1: Travel Time Distributions for  $P_A$  and  $P_B$

Travel time (mins)	40	50	60	70	AVG
$P_A$	0.5	0.2	0.2	0.1	49
$P_B$	0	0.8	0.2	0	52

Travel cost distributions of an uncertain road networks are often built from available vehicle trajectory data. Cost distributions are used on two model settings: in the classical *edge-centric* (EDGE) model [22, 23, 35] and in the recent *PAth-Centric* (PACE) model [4,

<sup>1</sup><https://www.postnord.dk/>

<sup>2</sup><https://flexdanmark.dk/>

33]. In the paper, we study the problem of arriving on time in PACE uncertain road networks. This is proven to be a more accurate way to model travel costs in real traffic [4, 33].

Assume that 100 trajectories occurred on path  $P = \langle e_1, e_2 \rangle$ , where 80 trajectories spent 10 minutes on  $e_1$  and  $e_2$ , respectively; and the other 20 trajectories spent 15 minutes each on both  $e_1$  and  $e_2$ . These trajectories suggest strong dependency—a driver is either fast or slow on both edges. A driver is unlikely to be fast on one edge and slow on the other edge. However, the EDGE model ignores such dependency and simply splits the trajectories to fit edges  $e_1$  and  $e_2$ , and both edges are assigned the distribution  $\{[10, 0.8], [15, 0.2]\}$ , meaning that traversing  $e_1$  or  $e_2$  takes 10 minutes with probability 0.8 and 15 minutes with probability 0.2.

The PACE model solves the problem by maintaining the “correct” joint distributions for paths directly. Specifically, following the above example, in addition to the edge distributions, the PACE model also maintains the joint distribution that captures, e.g., 10 and 10 minutes on  $e_1$  and  $e_2$  with a probability of 0.8, derived from the original, non-split trajectories (see Table 2(a)), which preserves cost dependencies.

Based on this joint distribution, the derived cost distribution of path  $P$  is consistent with the original trajectories (see Table 2(b)). However, when using the PACE model, it is unlikely that there will be sufficient data to maintain distributions for all meaningful paths in the underlying road network. In particular, the longer a path is, the fewer trajectories occurred on the path. Thus, the PACE model resorts to maintaining distributions for short paths with sufficient amounts of trajectories, e.g., above a threshold. We call such paths *trajectory-paths* or *T-paths* for short.

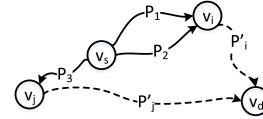
**Table 2: The Path-Centric Model, PACE**

(a) Joint Distribution $W_j(P)$		(b) Cost Distribution $D(P)$	
$\langle e_1, e_2 \rangle$	Probability	$P$	Probability
10, 10	0.80	$10 + 10 = 20$	0.80
15, 15	0.20	$15 + 15 = 30$	0.20

Computing the cost distribution of a path in PACE is complicated by the existence of multiple ways of computing the cost from the distributions of different T-paths and edges. For example, assume that distributions for edges  $e_1$ ,  $e_2$ , and  $e_3$  and for T-paths  $\langle e_1, e_2 \rangle$  and  $\langle e_2, e_3 \rangle$  exist. When computing the distribution of path  $\langle e_1, e_2, e_3 \rangle$ , we can use the distributions for T-path  $\langle e_1, e_2 \rangle$  and edge  $e_3$ , edge  $e_1$  and T-path  $\langle e_2, e_3 \rangle$ , or T-path  $\langle e_1, e_2 \rangle$  and T-path  $\langle e_2, e_3 \rangle$  which overlap. An existing study [4] proves that the last option provides the best accuracy as T-path  $\langle e_1, e_2 \rangle$ 's distribution preserves the dependency between  $e_1$  and  $e_2$  and T-path  $\langle e_2, e_3 \rangle$ 's distribution preserves the dependency between  $e_2$  and  $e_3$ .

Although the PACE model offers better accuracy than the EDGE model, stochastic routing on the PACE model often takes longer than the EDGE model [1, 33], which is mainly due to two challenges. **Search Heuristics:** Stochastic routing often needs to explore many candidate paths from the source to intermediate vertices. The efficiency depends highly on whether an algorithm is able to explore more promising paths before less promising ones. To enable this,

the existing solution in the PACE model considers the distributions of the candidate paths themselves [33]. However, it ignores the cost from the intermediate vertices to the destination, thus often leading to the exploration of non-promising paths.



**Figure 1: Motivating Example**

For example, in Figure 1, assume that we have three candidate paths  $P_1$ ,  $P_2$ , and  $P_3$  from source  $v_s$ . When only considering the candidate paths themselves,  $P_3$  is the most promising, i.e., having the least expected cost and the largest probability that the cost is below the budget. However,  $P_3$  is a bad choice since  $v_j$  is far away from destination  $v_d$ .

If we also consider the cost from the intermediate vertex to the destination,  $P_1$  and  $P_2$  become more promising because although  $P_1$  and  $P_2$  themselves have higher cost than  $P_3$ , intermediate vertex  $v_i$  is closer to the destination. Thus, heuristics that are able to accurately estimate the cost of travel from any intermediate vertex to the destination can be very effective, assuming that the heuristics are admissible to ensure correctness. We propose two types of admissible heuristics, i.e., binary and budget-specific, for the PACE model that are able to estimate the maximum probability of arriving at the destination within the budget from an intermediate vertex.

**Effective Pruning:** Efficient stochastic routing in the EDGE model relies on stochastic dominance-based pruning [22, 23]. For example, consider Figure 1, where candidate paths  $P_1$  and  $P_2$  both go from the source  $v_s$  to an intermediate vertex  $v_i$ . If the cost distribution of  $P_1$  stochastically dominates the cost distribution of  $P_2$ , we can safely prune  $P_2$ . This is equivalent to the deterministic case where  $P_1$  has a smaller cost value than  $P_2$ . More specifically, due to the independence assumption in the EDGE model, no matter which edge  $e$  is extended from  $v_i$ , it is guaranteed that  $D(P_1) \oplus D(e)$  stochastically dominates  $D(P_2) \oplus D(e)$  [22, 23], where  $\oplus$  represents the convolution operator and  $D(\cdot)$  represents the cost distribution of a path or an edge. Thus, there is no need to further consider  $P_2$  because there is always a better path that is extended from  $P_1$ .

Since the PACE model does consider dependency and does not use convolution to compute the distribution of a path, such pruning is no longer valid. As a result, massive amounts of candidate paths must be explored, which adversely affects efficiency. We propose virtual-path based stochastic routing for PACE as the second speedup technique. Here, we iteratively combine overlapping T-paths into so-called virtual paths (V-paths for short), such that time consuming distribution combination among overlapping T-paths is pre-computed before routing. Further, as the V-paths preserve the dependencies in the T-paths, it is possible to use stochastic dominance based pruning with the V-paths, which improves efficiency.

To summarize, we make three contributions in the paper. First, we propose two heuristics that are able to estimate the costs of travel from intermediate vertices to the destination in the PACE model. Second, we propose a method that introduces V-paths in the PACE model, which enables effective pruning. Third, we conduct

comprehensive experiments using two real-world data sets to gain detailed insight into the effectiveness of the speedup techniques.

The paper is organized as follows. Section 2 covers the basics of the PACE model. Section 3 presents the two versions of search heuristics followed by virtual-path based stochastic routing in Section 4. Section 5 offers the experimental studies, Section 6 reviews related work, and Section 7 concludes. We provide additional information elsewhere [8].

## 2 PRELIMINARIES

We cover the *path*-centric models and define the problem.

### 2.1 Path-Centric Model (PACE)

The path-centric model (PACE) [4, 33] models a road network as a directed graph  $\mathcal{G}^P = (\mathbb{V}, \mathbb{E}, \mathbb{P}, \mathbb{W})$ , where  $\mathbb{V}$  and  $\mathbb{E}$  still represent vertices and edges. Next,  $\mathbb{P}$  is a set of paths that have been traversed by at least  $\tau$  trajectories. We call these trajectory paths (a.k.a., T-paths). We evaluate the impact of threshold  $\tau$  in experiments. Finally,  $\mathbb{W}$  is a weight function set. The first weight function  $W : \mathbb{E} \rightarrow \mathbb{D}$  returns a cost distribution for each edge  $e \in \mathbb{E}$ , as in the EDGE model. For each T-path, we maintain weight functions  $W_J$  and  $W$ , both with signature  $\mathbb{P} \rightarrow \mathbb{D}$ . Given a T-path,  $W_J$  returns its joint distribution that models the cost dependency among the edges in the T-path, e.g., as seen in Table 2(a); and  $W$  returns a distribution that captures the total cost of the T-path, e.g., as seen in Table 2(b).

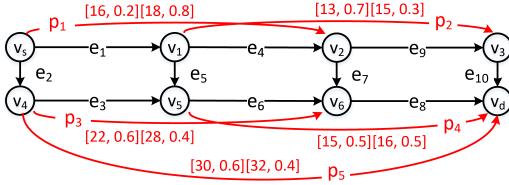


Figure 2: Path-Centric Uncertain Road Network, PACE

Figure 2 shows a PACE graph with 5 T-paths in red lines and the road network in black lines.

Due to the space limitation, Figure 2 only shows the total cost distributions of T-paths and omits the joint distributions. Next, we sketch how to compute the distribution of a path  $P = \langle e_1, e_2, \dots, e_n \rangle$  in PACE. As illustrated in the introduction, this distribution can generally be assembled from different sets of edge and T-path costs. It has been shown that the coarsest combination, i.e., the combination with the longest overlapping T-paths, gives the most accurate uncertain travel time [4]. For example, when computing the distribution of  $P = \langle e_1, e_4, e_9 \rangle$ , the coarsest combination is  $\{p_1, p_2\}$ , which is coarser than combinations  $\{p_1, e_9\}$ ,  $\{e_1, p_2\}$ , and  $\{e_1, e_4, e_9\}$ . After identifying the coarsest T-path sequence  $CPS(P) = (p_1, p_2, \dots, p_m)$  for path  $P$ , we use a T-path assembly operation  $\diamond$  to compute the joint distribution of  $P$  [33]:

$$D_J(P) = p_1 \diamond p_2 \diamond \dots \diamond p_m = \frac{\prod_{i=1}^m W_J(p_i)}{\prod_{i=1}^{m-1} W_J(p_i \cap p_{i+1})}, \quad (1)$$

where  $p_i \cap p_{i+1}$  denotes the overlap sub-path of the T-paths  $p_i$  and  $p_{i+1}$ . From the joint distribution maintained in PACE, we can derive the distribution of the cost of the path  $D_J(P)$  that captures

cost dependencies. Consider the example of  $P = \langle e_1, e_4, e_9 \rangle$ . In the PACE model, we have  $D_J(P) = \frac{W_J(p_1) \cdot W_J(p_2)}{W_J(\langle e_4 \rangle)}$  as  $p_1 \cap p_2 = \langle e_4 \rangle$ . In the case of no overlapping between two sequential T-paths, they are assembled by a convolution operation, as implemented in the EDGE.

We make a note on the notation in the paper: (1)  $W_J$  and  $W$  represent distributions that are maintained in EDGE or PACE; (2)  $D_J$  and  $D$  represent distributions that are derived from  $W_J$  and  $W$ ; (3)  $W_J$  and  $D_J$  indicate joint distributions.

Finally, EDGE and PACE each supports time-dependent uncertainty modeling by maintaining different uncertain graphs for different time periods, e.g., peak vs. off-peak hours.

### 2.2 Problem Definition

Given a source vertex  $v_s$ , a destination vertex  $v_d$ , a departure time  $t$ , and a travel cost budget  $B$ , we aim at identifying path  $P^*$  from  $v_s$  to  $v_d$  that has the largest probability of arriving at destination  $v_d$  within the budget  $B$  when leaving at  $t$ .

$$P^* = \arg \max_{P \in \text{Path}} \text{Prob}(D(P) \leq B), \quad (2)$$

where set *Path* contains all paths from  $v_s$  to  $v_d$ , and  $D(P)$  refers to the cost distribution of path  $P$  leaving at  $t$  using the PACE model. For example, given travel time budget  $B = 60$  minutes, we aim at finding the path that has the highest probability of arriving  $v_d$  within 60 minutes, if a driver departs at 8:00 a.m.

**Limitations of existing stochastic routing in PACE:** The Existing stochastic routing in PACE [33] (available in [8]) faces the two challenges. First, it starts exploring candidate paths from source  $v_s$ , each time extending a candidate path with its neighboring edges. A priority queue is used to maintain candidate paths, and each candidate path has a priority, e.g., the expected cost of the candidate path or the probability that the candidate path has cost below the budget  $B$ . It keeps exploring candidate paths until no path with higher probability exists. This approach determines the candidate path exploration priority purely based on the distributions of different candidate paths, while ignoring the costs from intermediate vertices to the destination. This leads to unnecessary exploration of non-competitive paths (e.g.,  $P_3$  in Figure 1), yielding reduced efficiency.

Second, stochastic dominance based pruning is inapplicable in the PACE model. Consider  $P_1$  and  $P_2$  in Figure 1. When extending both  $P_1$  and  $P_2$  with edge  $e$  to get two new paths  $P'_1$  and  $P'_2$ ,  $D(P_1) \diamond p_x$  and  $D(P_2) \diamond p_y$  are the distributions for  $P'_1$  and  $P'_2$ . Here,  $p_x$  and  $p_y$  are the longest T-paths that cover  $e$  and some edges in  $P_1$  and  $P_2$ , respectively, and  $p_x$  and  $p_y$  are often different. Thus, although  $D(P_1)$  stochastically dominates  $D(P_2)$ ,  $D(P_1) \diamond p_x$  may not stochastically dominate  $D(P_2) \diamond p_y$ , as  $p_x$  and  $p_y$  may be different, rendering stochastic dominance based pruning inapplicable.

## 3 SEARCH HEURISTICS

### 3.1 Definition of Heuristics

A routing algorithm explores a search space of candidate paths. During the exploration, it is beneficial to first explore paths that are expected to have high probabilities of arriving at the destination within the budget. Assume that we explore a path  $P_i$  from source

$v_s$  to an intermediate vertex  $v_i$ . We are able to compute the cost distribution of  $P_i$  using the PACE model. However, we do not know the cost distribution of the path from intermediate vertex  $v_i$  to the destination  $v_d$  because we do not yet know that path—there may possibly be many paths from  $v_i$  to  $v_d$ .

We target a heuristic function  $U(v_i, v_d, x)$  that is able to estimate the largest possible probability, among possibly multiple paths from  $v_i$  to  $v_d$ , of having a cost within  $x$  units. Supported by the branch-and-bound principle that is often used to solve optimization problems with admissible heuristics [6], we need to ensure that the heuristic function is *admissible*, meaning that the function never *underestimates* the probability of being able to travel from  $v_i$  to  $v_d$  within  $x$  units. In other words, the heuristic function must provide an *upper bound* on the probability.

If the upper bound probability of a candidate path is lower than the probability of a path  $P^*$  that already reaches the destination, then the candidate path can be pruned safely because it cannot get a higher probability than that of  $P^*$ . This guarantees correctness.

Since the destination is fixed for a given query, we often omit  $v_d$  and use  $U(v_i, x)$  when the query is clear. However, the heuristic function is destination-specific, meaning that  $U(v_i, x)$  is different for different query destinations. Based on the heuristic function, we are able to estimate the maximum probability of arriving at destination  $v_d$  within budget  $B$  when using path  $P_i$  as follows:

$$\maxProb(P_i, B) = \sum_{t=1}^B D(P_i).pdf(t) \cdot U(v_i, B-t), \quad (3)$$

where  $D(P_i).pdf(t)$  denotes the probability of the travel taking  $t$  units when following path  $P_i$  from source  $v_s$  to intermediate vertex  $v_i$ , which is computed using the PACE model. Next,  $U(v_i, B-t)$  denotes the largest probability of travel from  $v_i$  to  $v_d$  costing at most  $B-t$  units, which is returned by the heuristic function.

The maximum probability of arriving at the destination within  $B$  units when using path  $P_i$  is the sum, over all possible costs  $t$ , of the products of the probability that  $P_i$  takes  $t$  units and the maximum probability that the remaining path from  $v_i$  to  $v_d$  takes at most  $B-t$  units, as illustrated in Figure 3(a). In Figure 3(b),  $P_i$ 's distribution is  $\{[8, 0.9], [10, 0.1]\}$ . When budget  $B = 25$ , we get  $\maxProb(P_i, 25) = 0.9 \cdot U(v_i, 17) + 0.1 \cdot U(v_i, 15)$ .

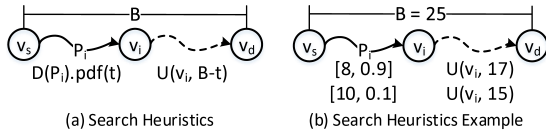


Figure 3: Search Heuristics

With the heuristic function, we are able to first explore more promising paths, i.e., paths that have higher  $\maxProb$  values. In addition, it enables early stopping. If the path with the largest  $\maxProb$  value, say  $P^*$ , already reaches the destination, we can safely stop exploring other paths. This is because, even when using admissible heuristics, the other paths can only achieve probabilities that are no greater than  $P^*$ 's probability. Thus, in the best case, the other paths cannot have a larger probability than that of  $P^*$  and it is safe to stop.

We proceed to introduce two different strategies for instantiating the admissible heuristic function  $U(v_i, x)$ —a binary heuristic and a budget-specific heuristic.

### 3.2 Binary Heuristic

The binary heuristics simplifies function  $U(v_i, x)$  by only distinguishing between probability values 0 and 1—whether or not it is possible to go from  $v_i$  to the destination within  $x$  units.

To this end, we maintain an auxiliary function  $v_i.getMin()$  for each vertex  $v_i$  that returns the *least travel cost* required to travel from  $v_i$  to destination  $v_d$ . We define the binary heuristics in Eq. 4.

$$U(v_i, x) = \begin{cases} 0 & \text{if } x < v_i.getMin() \\ 1 & \text{if } x \geq v_i.getMin() \end{cases} \quad (4)$$

If  $x$  is below  $v_i.getMin()$ , it is impossible to go from  $v_i$  to  $v_d$  within  $x$  units. As  $v_i.getMin()$  is the smallest travel cost within which it is possible to reach  $v_d$ , it is impossible to reach  $v_d$  with a travel cost  $x$  that is smaller than  $v_i.getMin()$ . Thus, the highest probability of reaching  $v_d$  within  $x$  units is 0. Otherwise, if  $x \geq v_i.getMin()$ , it is possible to reach  $v_d$  within  $x$  units. In this case, we set  $U(v_i, x) = 1$ , which is a very optimistic estimate. The resulting binary heuristic is *admissible* because the probability of reaching  $v_d$  within  $x$  units is at most 1, meaning that the heuristic never underestimates the probability. In other words, the actual probability of reaching  $v_d$  from  $x_i$  may be less than 1 because it can also take more than  $x$  units.

Next, we instantiate function  $v_i.getMin()$  for a given destination  $v_d$ . The main idea is to perform a backward search from destination  $v_d$  to all other vertices and then annotate each vertex with the least travel cost from the vertex to  $v_d$ .

To this end, we first build a reversed graph  $\mathcal{G}_{rev}^P = (\mathbb{V}, \mathbb{E}', \mathbb{P}', \mathbb{W}')$  from  $\mathcal{G}^P = (\mathbb{V}, \mathbb{E}, \mathbb{P}, \mathbb{W})$ , where the vertices remain unchanged but the directions of the edges in  $\mathbb{E}'$  and T-paths in  $\mathbb{P}'$  are reversed compared to the counterparts in  $\mathbb{E}$  and  $\mathbb{P}$ . Weight function  $\mathbb{W}'$  maintains only deterministic costs, thus mapping a reversed edge or T-path to the minimum cost in its distribution. For example, Figure 5 includes a reversed edge  $e'_1 = (v_1, v_s)$  has cost  $\mathbb{W}'(e'_1) = 8$  because of edge  $e_1 = (v_s, v_1)$  in  $\mathcal{G}^P$  (cf. black part of 2) that has cost distribution  $\mathbb{W}(e_1) = \{[8, 0.9], [10, 0.1]\}$ , in which 8 is the minimum cost. Similarly, T-path  $p_1 = (e_1, e_4)$  in Figure 2 yields the reversed T-path  $p'_1 = (e'_4, e'_1)$  in Figure 5, with  $\mathbb{W}'(p'_1) = 16$ .

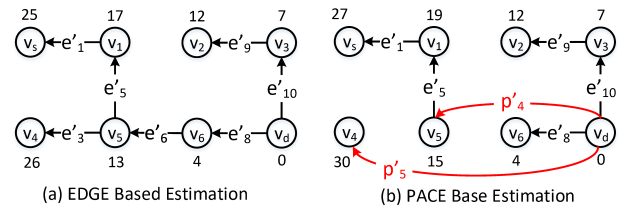


Figure 4: Computing  $v_i.getMin()$ , Given Destination  $v_d$

Next, we explore the reversed graph  $\mathcal{G}_{rev}^P$ , starting from  $v_d$ , annotating each vertex with the least cost from  $v_d$  to the vertex. If only considering reversed edges and disregarding reversed T-paths [1],

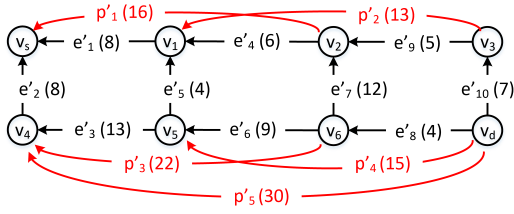


Figure 5: Reversed Graph  $\mathcal{G}_{rev}^p$

Dijkstra's algorithm could produce a shortest path tree, as exemplified in Figure 4(a). However, the reversed T-paths render Dijkstra's algorithm inapplicable.

For example, when only considering reversed edges, the vertex  $v_5$  is annotated with 13 (i.e., the sum of the weights of  $e'_8$  and  $e'_6$ ). When considering reversed T-paths, reversed T-path  $p'_4$  has a weight  $W'(p'_4) = 15$ , which exceeds the sum of the weights of  $e'_8$  and  $e'_6$ . According to the theory underlying the PACE model [4, 33], the T-path weight is more accurate as it preserves the dependency between the two edges better. Thus, we should annotate  $v_5$  with the reversed T-path cost  $W'(p'_4) = 15$ . However, if we simply run Dijkstra's algorithm,  $v_5$  is annotated with 13 as it is smaller than 15. This exemplifies why we need a new algorithm that is able to consider reversed T-paths that produce more accurate weights in PACE.

Algorithm 1 shows a new shortest path tree generation algorithm that takes into account the weights of the T-paths. We maintain two costs  $c_1$  and  $c_2$  for each vertex  $v_i$ . Cost  $c_1$  represents the shortest path weight from  $v_d$  to  $v_i$ , where the shortest path may use both reversed edges and reversed T-paths. Cost  $c_2$  represents the number of edges in the reversed T-paths that are used to connect  $v_d$  to  $v_i$ . We prefer low values for  $c_1$  and high values for  $c_2$ , meaning that we prefer shortest paths and reversed T-paths with many edges. We use a priority queue  $Q$  to maintain the vertices, where the priority is according to cost  $c_1$ . Table 3 shows the first three iterations when applying Algorithm 1 to  $\mathcal{G}_{rev}^p$ , from Figure 5. In the first three iterations, the priority queue peeks vertices  $v_d$ ,  $v_6$ , and  $v_3$ , respectively, as shown in the top row of Table 3. For each iteration, we show the vertices in the priority queue  $Q$ , the costs  $c_1$  and  $c_2$  of each vertex, and the reversed edge or T-path that connects the vertex from its parent in the shortest path tree, denoted by *Par*.

Starting from  $v_d$ , its neighbor vertices  $v_6$ ,  $v_3$ ,  $v_5$ , and  $v_4$  are explored by following reversed edges  $e'_8$  and  $e'_{10}$  and reversed T-paths  $p'_4$  and  $p'_5$ , respectively (Line 9), resulting in  $c_1$  values 4, 7, 15, and 30 (Line 10). Since vertices  $v_5$  and  $v_4$  are reached from  $v_d$  via reversed T-paths  $p'_4 = \langle e'_8, e'_6 \rangle$  and  $p'_5 = \langle e'_8, e'_6, e'_3 \rangle$ , their  $c_2$  values are 2 and 3, indicating that there are 2 and 3 edges in the reversed T-paths that connect  $v_d$  to  $v_5$  and to  $v_6$ , respectively (using function *countEdge(ep)* in Line 11). The  $c_2$  values for  $v_6$  and  $v_3$  are zero because they do not use reversed T-paths.

In the second iteration, as shown in Table 3, the priority queue peeks  $v_6$ , which has three neighbors  $v_5$ ,  $v_2$ , and  $v_4$  by following  $e'_6$ ,  $e'_7$ , and  $p'_3$ . When visiting  $v_5$  by the reversed edge  $e'_6$ , we obtain  $\hat{c}_1 = 13$  and  $\hat{c}_2 = 0$  (Lines 10 and 11). Now, we need to consider whether we should replace the existing  $c_1$  and  $c_2$  values for vertex  $v_5$  in the priority queue with  $\hat{c}_1$  and  $\hat{c}_2$ .

### Algorithm 1: Shortest Path Tree Generation in PACE

---

**Input:**  $\mathcal{G}_{rev}^p = (\mathbb{V}, \mathbb{E}', \mathbb{P}', \mathbb{W}')$ ,  $v_d$ ;  
**Output:** Shortest path tree  $T$  from  $v_d$ ;

- 1 *Init*( $\mathbb{V}$ ); /\* set  $c_1 \leftarrow \infty$  and  $c_2 \leftarrow 0$  for each vertex  $*$  \*/
- 2 Shortest path tree  $T \leftarrow \emptyset$ ;
- 3 Priority queue  $Q \leftarrow \emptyset$ ;
- 4  $v_d.c_1 \leftarrow 0$ ;
- 5  $Q.push(v_d)$ ;
- 6 **while**  $Q \neq \emptyset$  **do**
- 7      $v \leftarrow Q.peek()$ ;
- 8      $T.add(v)$ ;
- 9     **for each** vertex  $u$  that is reachable from  $v$  by a reversed edge or T-path  $ep$  **do**
- 10          $\hat{c}_1 \leftarrow v.c_1 + W'(ep)$ ;
- 11          $\hat{c}_2 \leftarrow v.c_2 + countEdges(ep)$ ;
- 12         **switch** *checkDominance*( $\hat{c}_1, \hat{c}_2, u.c_1, u.c_2$ ) **do**
- 13             **case** *NON-DOMINATION* **do**
- 14                  $\hat{P}_{old} \leftarrow tracePath(v_d, u)$ ;
- 15                  $\hat{P}_{new} \leftarrow tracePath(v_d, v) + \langle v, u \rangle$ ;
- 16                 **if** ( $\hat{P}_{old} \neq \hat{P}_{new}$ ) && ( $\hat{c}_1 < u.c_1$ ) **then**
- 17                      $u.c_1 \leftarrow \hat{c}_1$ ;  $u.c_2 \leftarrow \hat{c}_2$ ;  $u.parent \leftarrow v$ ;
- 18                      $Q.add(u)$ ;
- 19                 **if** ( $\hat{P}_{old} == \hat{P}_{new}$ ) && ( $\hat{c}_2 > u.c_2$ ) **then**
- 20                      $u.c_1 \leftarrow \hat{c}_1$ ;  $u.c_2 \leftarrow \hat{c}_2$ ;  $u.parent \leftarrow v$ ;
- 21                      $Q.add(u)$ ;
- 22             **case** *DOMINATION* **do**
- 23                  $u.c_1 \leftarrow \hat{c}_1$ ;  $u.c_2 \leftarrow \hat{c}_2$ ;  $u.parent \leftarrow v$ ;
- 24                  $Q.add(u)$ ;
- 25 **return**  $T$ ;

---

Table 3: Shortest Path Tree Generation in PACE

Iteration 1			Iteration 2			Iteration 3		
$v_d.c_1 = 0$			$v_6.c_1 = 4$			$v_3.c_1 = 7$		
$Q$	$c_1, c_2$	<i>Par</i>	$Q$	$c_1, c_2$	<i>Par</i>	$Q$	$c_1, c_2$	<i>Par</i>
$v_6$	4, 0	$e'_8$	$v_3$	7, 0	$e'_{10}$	$v_5$	15, 3	$p'_4$
$v_3$	7, 0	$e'_{10}$	$v_5$	15, 2	$p'_4$	$v_2$	12, 0	$e'_9$
$v_5$	15, 2	$p'_4$	$v_2$	16, 0	$e'_7$	$v_1$	20, 2	$p'_2$
$v_4$	30, 3	$p'_5$	$v_4$	30, 3	$p'_5$	$v_4$	30, 3	$p'_5$

To utilize the accurate cost values maintained in reversed T-paths, a smaller  $c_1$  but a larger  $c_2$  are preferred, meaning that we aim at (1) identifying the least cost from  $v_d$  to a vertex and at (2) maximizing the use of reversed T-paths, especially long ones, during the search.

Following this principle, we need to consider both values when making a decision on whether to update an existing vertex in the priority queue based on a new candidate path. More specifically, vertex  $v_5$  has  $c_1 = 15$  and  $c_2 = 2$  through  $\langle p'_4 \rangle$  in the priority queue. Now, the new candidate path  $\langle e'_8, e'_6 \rangle$  has  $\hat{c}_1 = 13$  and  $\hat{c}_2 = 0$ . We apply a *checkDominance*( $\cdot$ ) function (Line 12) to compare the two values following the concept of pareto-optimality. We define two relationships—NON-DOMINATION and DOMINATION.

NON-DOMINATION (lines 13–21). If the new candidate path is better in one value but worse in the other than those already in the

priority queue, we are in the non-domination case. For example, since  $\hat{c}_1 < v_5.c_1$  and  $\hat{c}_2 < v_5.c_2$ , this is a non-domination case.

In the non-domination case, we consider the new candidate path  $\hat{P}_{new}$  and the old path  $\hat{P}_{old}$  that leads to the values in the priority queue. We check whether they correspond to the same path in the road network (using function  $tracePath(\cdot)$  in lines 14–15). If yes, following criterion (2), we use the path with more edges covered by T-paths as the corresponding cost should be more accurate. If no, we follow criterion (1) and use the path that has a smaller  $c_1$  value to make sure that we find the minimum cost of candidate paths.

In the above example, paths  $\hat{P}_{new}$  and  $\hat{P}_{old}$  both correspond to the same path  $\langle e'_5, e'_6 \rangle$ . We then keep the values in the priority queue since  $v_5.c_2 = 2$  indicates that two edges are covered by T-paths, while  $\hat{c}_2 = 0$  indicates that no edges are covered by T-paths in the new candidate path  $\hat{P}_{new}$ .

**DOMINATION** (Lines 22–24). If the candidate path is better in one value and not worse in the other value than those already in the priority queue, domination occurs. In the domination case, we update the priority queue using the candidate path.

For example, at the third iteration, where  $v_3$  is peeked from the priority queue,  $v_2$  is visited again as  $v_3$ 's neighbor via  $e'_3$ . We now have  $v_2.c_1 = 16$  and  $v_2.c_2 = 0$ , and  $\hat{c}_1 = 12$  and  $\hat{c}_2 = 0$ . Since  $\hat{c}_1 < v_2.c_1$  and  $\hat{c}_2 = v_2.c_2$ , we update  $v_2.c_1$  and  $v_2.c_2$  to be  $\hat{c}_1$  and  $\hat{c}_2$ . If the candidate path is dominated by the values already in the priority queue, we do nothing.

Figure 4(b) shows the identified shortest path tree using both reversed edges and T-paths.

The time and space complexity of Algorithm 1 is  $O((|\mathbb{E}'| + |\mathbb{P}'|)lg|\mathbb{V}|)$  and  $O(|\mathbb{V}|^2)$ , respectively. Analysis details are provided elsewhere [8].

### 3.3 Budget-Specific Heuristic

The binary heuristic often provides overly optimistic probabilities from an intermediate vertex to the destination, which can lead to unnecessary exploration of paths that cannot provide high probabilities of arriving at the destination within the budget. To enable more accurate heuristics, we propose a budget-specific heuristic that formulates function  $U(v_i, x)$  at a finer granularity so that it can return probabilities in the range  $[0, 1]$ . Specifically,

$$U(v_i, x) = \max_{z \in ON(v_i)} H(v_i, z, x),$$

$$H(v_i, z, x) = \sum_{k=1}^x \mathbb{W}(\langle v_i, z \rangle).pdf(k) \cdot U(z, x - k), \quad (5)$$

and  $ON(v_i)$  includes  $v_i$ 's outgoing neighbor vertices. Recall that the heuristic function  $U(v_i, x)$  returns the largest probability of reaching  $v_d$  from  $v_i$  within  $x$  units. Eq. 5 computes this probability by getting the largest  $H(v_i, z, x)$  over all  $v_i$ 's outgoing neighbor vertices  $z$ , where  $H(v_i, z, x)$  represents the probability of arriving at  $v_d$  within time budget  $x$  when following  $\langle v_i, z \rangle$  by first going from  $v_i$  to vertex  $z$  and then to  $v_d$ . Note that  $\langle v_i, z \rangle$  may be an edge or a T-path. Since the heuristic function never ignores any possibility leaving from  $v_i$ , the function never underestimates the probability and thus is *admissible*. More specifically, we define  $H(v_i, z, x)$  as the sum, over all possible costs  $k \in [1, x]$ , of the products of the probability that it takes  $k$  units from  $v_i$  to vertex

$z$ , i.e.,  $\mathbb{W}(\langle v_i, z \rangle).pdf(k)$ , and the probability that it takes at most  $x - k$  units to travel from  $z$  to destination  $v_d$ , i.e.,  $U(z, x - k)$ .

**3.3.1 Heuristic Tables.** As we aim at providing  $U(v_i, x)$  at a fine granularity, we need to consider different values for  $x$ . To this end, we define the finest budget granularity  $\delta$  and then consider multiple budget values  $\delta, 2 \cdot \delta, \dots$ , and  $\eta \cdot \delta$  for  $x$ . The largest value  $\eta \cdot \delta$  ensures that  $U(v_i, \eta \cdot \delta) = 1$  for all vertices  $v_i$  given a destination vertex  $v_d$ . Then, we can represent function  $U(v_i, x)$  as a *heuristic table* with  $|\mathbb{V}|$  rows and  $\eta$  columns. The cell in the  $i$ -th row and  $j$ -th column represents value  $U(v_i, j \cdot \delta)$ .

Table 4 shows an example heuristic table for destination vertex  $v_d$  based on the graph shown in Figure 2. Here,  $\delta$  is set to 3, and the largest budget is 36 because no matter at which vertex we start, the probability that we reach destination  $v_d$  within 36 units is 1. The blue cell in the row of vertex  $v_5$  and the column of budget  $x = 15$  denotes  $U(v_5, 15) = 0.5$ , meaning that the largest probability of reaching  $v_d$  from  $v_5$  within time budget 15 is 0.5.

**Table 4: Heuristic Table of  $U(v_i, x)$ , Destination  $v_d$**

x	3	6	9	12	15	18	21	24	27	30	33	36
$v_5$	0	0	0	0	0	0	0	0	0	0.18	0.91	1
$v_1$	0	0	0	0	0	0	0.12	1	1	1	1	1
$v_2$	0	0	0	0	0.6	1	1	1	1	1	1	1
$v_3$	0	0	1	1	1	1	1	1	1	1	1	1
$v_4$	0	0	0	0	0	0	0	0	0	0.6	1	1
$v_5$	0	0	0	0	0.5	1	1	1	1	1	1	1
$v_6$	0	1	1	1	1	1	1	1	1	1	1	1
$v_d$	1	1	1	1	1	1	1	1	1	1	1	1

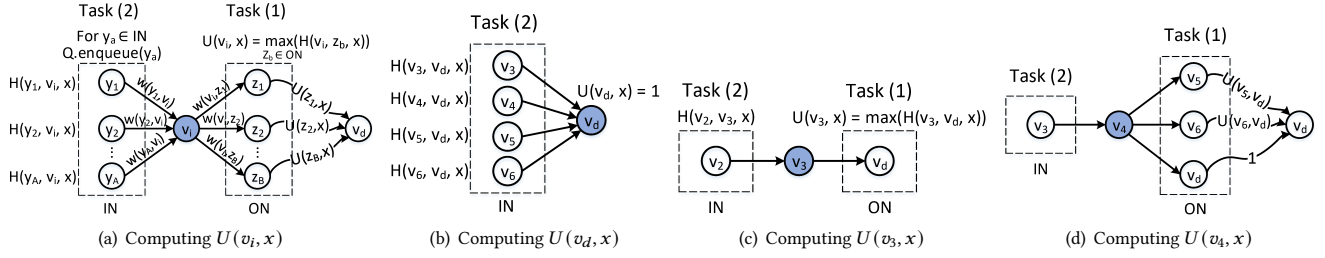
The binary heuristic can also be represented as a heuristic table. Specifically, for vertex  $v_i$ , we identify the smallest budget value in the table that is larger than or equal to  $v_i.getMin()$ , say  $j \cdot \delta$ . Then, in the row for  $v_i$ , the first  $j - 1$  cells are 0, and the remaining cells are 1.

The budget-specific heuristic provides finer-granularity probabilities than the binary heuristic, which estimates the probability of an intermediate vertex  $v_i$  reaching  $v_d$  within the time budget more accurately. Thus, budget-specific heuristic helps path exploration so that more promising candidate paths are explored first.

**3.3.2 Instantiating Heuristic Tables.** Efficiently instantiating a budget-specific heuristic table is non-trivial. A naive and computationally expensive approach is to compute according to Eq. 5 for each of  $|\mathbb{V}| \cdot \eta$  cells for each destination, i.e., for each of  $|\mathbb{V}|^2 \cdot \eta$  cells in the heuristic table.

We make two observations that can speed up the heuristic table instantiation. First, for each row,  $l$  is the smallest budget value whose cell is larger than 0, and  $s$  is the smallest budget values whose cell is 1. Thus, all budget values smaller than  $l$  should have cells of 0, and all budget values larger than  $s$  should have cells of 1. In other words, the cells to the left of  $l$ 's cell are 0, and the cells to the right of  $s$ 's cell are 1. For example, in Table 4, for  $v_5$ , we have  $l = 15$  and  $s = 18$ , and cells to the left of 15 are 0 and the cells on the right of 18 are 1. This observation suggests that for each vertex, we need to identify the two budget values  $l$  and  $s$  and compute values only for the cells between  $l$  and  $s$ .

Second, all cells for destination vertex  $v_d$  are 1, since the probability of going from the destination vertex to itself within any budget



**Figure 6: Computing Budget Specific Heuristic: (a) Provides an Abstraction of the Heuristic Computation for any Vertex  $v_i$ ; (b), (c) and (d) Show Concrete Heuristic Computations for Vertices  $v_d$ ,  $v_3$ , and  $v_4$ , Respectively**

is always 1. Thus  $U(v_d, x) = 1$  no matter what  $x$  is. This motivates us to compute according to Eq. 5 starting from the destination  $v_d$ .

Based on the above two observations, we propose Algorithm 2 to instantiate a heuristic table for a given destination  $v_d$ . Algorithm 2 takes as input graph  $\mathcal{G}^P$  and destination  $v_d$ , as well as  $\delta$  and  $\eta$  that specify different budget values. Algorithm 2 returns heuristic table  $\bar{U}$ , where each row includes the heuristic function values  $U(v_i, x)$  for a vertex, as exemplified by Table 4. To enable compact storage, we only store budget values between  $l$  and  $s$ . The search strategy in Algorithm 2 is shown in Figure 6.

We use a FIFO queue to maintain the vertices. First, we insert the destination  $v_d$ , and iteratively check each vertex in the queue until the queue is empty. For each iteration, we remove a vertex  $v_i$  from the queue and perform two tasks—(1) compute  $U(v_i, x)$  using Eq. 5 and the above two observations and (2) insert into the queue all incoming vertices that are connected to  $v_i$  by an edge or a T-path and that are not yet computed in the heuristic table (see Figure 6(a)).

### Algorithm 2: BudgetSpecificHeuristics

---

**Input:**  $\mathcal{G}^P = (\mathbb{V}, \mathbb{E}, \mathbb{P}, \mathbb{W})$ ,  $v_d$ ,  $\delta$ ,  $\eta$ ,  $\bar{U} \leftarrow \emptyset$ ;  
**Output:** Heuristic table  $\bar{U}$ ;

- 1 Queue  $Q \leftarrow \emptyset$ ;
- 2  $Q.enqueue(v_d)$ ;
- 3 **while**  $Q \neq \emptyset$  **do**
- 4    $v^* \leftarrow Q.dequeue()$ ; /\* Perform Task (1) \*/
- 5   **if**  $v^* = v_d$  **then**
- 6     **for each**  $i$  **from** 1 **to**  $\eta$  **do**
- 7        $\bar{U}(v^*, i \cdot \delta) \leftarrow 1$
- 8   **else**
- 9      $ComputeOneRowU(\mathcal{G}^P, v^*, \delta, \eta, \bar{U})$ ;
- 10   /\* Perform Task (2) \*/
- 11   **for each vertex**  $v$ , **where**  $\langle v, v^* \rangle \in \mathbb{E} \cup \mathbb{P}$  **do**
- 12     **if** The row for vertex  $v$  in  $\bar{U}$  is still empty **then**
- 13        $Q.enqueue(v)$ ;
- 14 **return**  $\bar{U}$ ;

---

**Task (1):** Based on the second observation, if  $v_i$  is the destination vertex  $v_d$ ,  $U(v_i, x) = 1$  for all budget value  $x$ . Otherwise, according to the first observation, we need to identify the two boundary budget values  $l$  and  $s$ . Here, the  $l$  value should be the smallest budget

### Algorithm 3: ComputeOneRowU

---

**Input:**  $\mathcal{G}^P = (\mathbb{V}, \mathbb{E}, \mathbb{P}, \mathbb{W})$ ,  $v_i$ ,  $\delta$ ,  $\eta$ ,  $\bar{U}$ ;

- 1  $l \leftarrow$  the smallest budget that is no less than  $v_i.getMin()$ ;
- 2  $s \leftarrow \eta \cdot \delta$ ;
- 3 **for each** each budget  $x$  that is smaller than  $l$  **do**
- 4    $\bar{U}(v_i, x) \leftarrow 0$ ; /\* left cells of  $l$  are all 0 \*/
- 5 /\* Compute  $U(v_i, x)$  using Eq. 5 \*/
- 6 **for each vertex**  $z$ , **where**  $\langle v_i, z \rangle \in \mathbb{E} \cup \mathbb{P}$ , **do**
- 7   **if** The row for vertex  $z$  in  $\bar{U}$  is still empty **then**
- 8      $ComputeOneRowU(\mathcal{G}^P, z, \delta, \eta, \bar{U})$ ;
- 9    $x \leftarrow l$ ;
- 10   **while**  $x < \eta \cdot \delta$  **do**
- 11      $H(v_i, z, x) \leftarrow 0$ ;
- 12     **for each cost**  $c \in \mathbb{W}(\langle v_i, z \rangle)$  **do**
- 13        $H(v_i, z, x) \leftarrow$
- 14        $H(v_i, z, x) + \mathbb{W}(\langle v_i, z \rangle).pdf(c) \cdot U(z, x - c)$ ;
- 15       **if**  $H(v_i, z, x) < 1$  **then**
- 16          $x \leftarrow x + \delta$ ; /\* next time budget \*/
- 17       **else**
- 18         **break**;
- 19    $s \leftarrow \min(s, x)$ ;
- 20 **for each budget**  $x \in [l, s]$  **do**
- 21    $\bar{U}(v_i, x) \leftarrow \max_{\langle v_i, z \rangle \in \mathbb{E} \cup \mathbb{P}} (H(v_i, z, x))$ ;
- 22 **for each budget**  $x > s$  **do**
- 23    $\bar{U}(v_i, x) \leftarrow 1$ ;

---

value that is no less than  $v_i.getMin()$  from the binary heuristics. This is because  $v_i.getMin()$  represents the least travel cost from  $v_i$  to  $v_d$ , and for any budget value that is smaller than  $v_i.getMin()$ , it is impossible to reach the destination within the budget. Then, we start computing  $U(v_i, x)$ , where  $x$  starts from  $l$  and each time increases by  $\delta$ . Once we find that  $U(v_i, x)$  is 1, we set the current  $x$  to  $s$  and the budgets that are larger than  $s$  to 1.

Then, we consider vertex set  $ON$  that includes  $v_i$ 's outgoing neighbor vertices (see Figure 6 (a)) to compute  $U(v_i, x)$  using Eq. 5:

$$\begin{aligned}
 U(v_i, x) &= \max_{z \in ON} H(v_i, z, x) \\
 &= \max_{z \in ON} \sum_{c=1}^x W(\langle v_i, z \rangle).pdf(c) \cdot U(z, x - c). \quad (6)
 \end{aligned}$$

The first term  $W(\langle v_i, z \rangle).pdf(c)$  is available from graph  $\mathcal{G}^P$ , no matter  $\langle v_i, z \rangle$  is an edge or a T-path. We use dynamic programming to compute the second term  $U(z, x - c)$ . As we start searching from the destination, vertex  $z$ 's row in the heuristic table often has already been computed, we may get  $U(z, x - c)$  directly from the table. Otherwise, we recursively apply the same equation to compute  $U(z, x - c)$ .

**Task (2):** We perform the second task using *IN* that includes  $v_i$ 's incoming neighbor vertices (see Figure 6 (a)). For each vertex  $y_a \in IN$ , we insert the vertex into the queue  $Q$ . In addition, we now can easily compute  $H(y_a, v_i, x) = \sum_{c=1}^x W(\langle y_a, v_i \rangle).pdf(c) \cdot U(v_i, x - c)$  since the first term is available from graph  $\mathcal{G}^P$  and since we have just computed  $U(v_i, x)$  in the first task.

Figure 6 (b), (c), and (d) illustrate the first three steps of computing the heuristic table for the graph shown in Figure 2 for destination  $v_d$ . The first step starts from destination  $v_d$  itself. Next, it explores  $v_d$ 's incoming neighbors  $v_3$  and  $v_4$ .

The time complexity of Algorithm 2 and 3 is  $O(|V| \cdot \eta^2 \cdot |out|)$ , where  $\eta$  is the number of columns in the heuristic table and  $|out|$  is the largest outdegree of a vertex. The space complexity is  $O(|V|^2 \cdot \eta)$ . The analysis details are provided elsewhere [8].

## 4 VIRTUAL PATH BASED ROUTING

Stochastic dominance based pruning is inapplicable in PACE because PACE captures cost dependencies and does not assume independence. More specifically, the cost distribution of a path is computed using the T-path assembly operation  $\diamond$  (c.f. Eq.1), not the convolution operation  $\oplus$ . To enable stochastic dominance based pruning, we propose to build V-paths, such that the cost distribution of a path can be computed by the convolution of cost distributions of V-paths, T-paths, and edges.

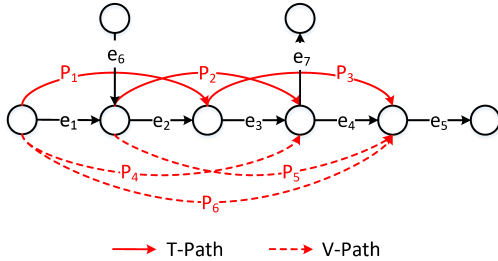


Figure 7: Example of Building V-paths from T-paths

### 4.1 Building Virtual Paths

We use a concrete example to illustrate the intuition of introducing V-paths. The example in Figure 7 includes T-paths  $p_1$ ,  $p_2$ , and  $p_3$ . Given a path  $P = \langle e_1, e_2, e_3, e_4, e_5 \rangle$ , its joint distribution is computed by  $D_J(P) = p_1 \diamond p_2 \diamond p_3 \diamond \langle e_5 \rangle$  according to Eq. 1. Then, we derive the cost distribution  $D(P)$  from the joint distribution using a simple transformation as shown in Table 2.

As  $p_3$  and  $\langle e_5 \rangle$  do not overlap, we have

$$D_J(P) = \frac{W_J(p_1)W_J(p_2)W_J(p_3)}{W(\langle e_2 \rangle)W_J(\langle e_3 \rangle)} W(e_5) = D_J(P')W(e_5), \quad (7)$$

where  $P' = \langle e_1, e_2, e_3, e_4 \rangle$  is a sub-path of  $P$ . This suggests that  $P'$  and  $e_5$  are independent.

Thus, we could apply the convolution operation to compute the cost distribution of  $P$ :  $D(P) = D(P') \oplus W(e_5)$ , where the cost distribution  $D(P')$  can be derived from the joint distribution of  $D_J(P')$ , i.e., the fraction part in Eq. 7, using T-paths  $p_1$ ,  $p_2$ , and  $p_3$ .

We call  $P'$  a *V-path*. First, a V-path must have fewer than  $\tau$  trajectories because, otherwise, it should already have been a T-path. Second, the distribution of a V-path needs to be computed from the T-path assembly operation  $\diamond$  (see Eq. 1) using distributions of multiple T-paths.

This example suggests that if we pre-compute the distribution of V-path  $P'$ , we are able to use only convolution  $\oplus$  to compute the distribution of  $P$ . This motivates us to systematically identify all V-paths and pre-compute their distributions. The intuition is that we move the time consuming online computation of Eq. 1 offline by pre-computing the distributions of all V-paths. As Eq. 1 is used both online and offline, there is no accuracy loss. This enables us to use only convolution to compute the distributions of candidate paths during routing, thus making it possible to use stochastic dominance based pruning in the PACE model.

The idea is to first combine overlapping T-paths into V-paths and then recursively combine overlapping V-paths into longer V-paths.

**Combining Two T-Paths:** In the first iteration, we combine overlapping T-paths into V-paths. If two T-paths overlap and the path underlying the two T-paths does not have a corresponding T-path, we combine the two T-paths into a V-path. For example, consider T-paths  $p_1$  and  $p_2$  in Figure 7 that overlap on edge  $e_2$ . The path underlying  $p_1$  and  $p_2$  is  $\langle e_1, e_2, e_3 \rangle$ , and no T-path exists that also corresponds to  $\langle e_1, e_2, e_3 \rangle$ . Thus, we build a V-path  $p_4$  with joint distribution  $D_J(p_4) = p_1 \diamond p_2$ . Similarly, we build V-path  $p_5$  with joint distribution  $D_J(p_5) = p_2 \diamond p_3$ .

**Combining Two V-Paths:** In the second iteration, we combine overlapping V-paths into longer V-paths. For example, in Figure 7, since the two V-paths  $p_4$  and  $p_5$  overlap, we build V-path  $p_6$ . The joint distribution of the new, longer V-path is also computed based on the corresponding T-paths. Here,  $D_J(p_6) = p_1 \diamond p_2 \diamond p_3$ . When combining two V-paths, we do not need to check if a T-path exists that corresponds to the same underlying path. The existence of the two V-paths implies that there are fewer than  $\tau$  trajectories. And since the V-paths are sub-paths of the combined path, the combined path also cannot have more than  $\tau$  trajectories. Thus, no T-path can exist for the combined path.

We keep combining overlapping V-paths to obtain longer V-paths until there are no more overlapping V-paths to combine. Next, we observe that we need all V-paths rather than only the longest ones. Table 5 shows that all V-paths  $p_4$ ,  $p_5$ , and  $p_6$  in Figure 7 contribute to computing the distribution of some path. So although we have a longer V-path  $p_6$ , we still need the two short V-paths  $p_4$  and  $p_5$ .

Table 5: Computing Path Distributions using Convolution

Path	Distribution
$\langle e_1, e_2, e_3, e_7 \rangle$	$D(p_4) \oplus W(e_7)$
$\langle e_6, e_2, e_3, e_4 \rangle$	$W(e_6) \oplus D(p_5)$
$\langle e_1, e_2, e_3, e_4, e_5 \rangle$	$D(p_6) \oplus W(e_5)$



Given graph  $\mathcal{G}^P = (\mathbb{V}, \mathbb{E}, \mathbb{P}, \mathbb{W})$ , the time and space complexity for building V-paths is  $O(|\mathbb{P}| \cdot |\mathbb{V}|)$  and  $O(|\mathbb{P}| \cdot |\mathbb{V}| \cdot |St|)$ , respectively, where  $|St|$  is the storage used for the longest V-path. Analysis details are found in [8].

**Updated PACE graph:** After generating all V-paths, we define an updated PACE graph  $\mathcal{G}^{P^+} = (\mathbb{V}, \mathbb{E}, \mathbb{P}^+, W^+)$ , where  $\mathbb{P}^+$  is a union of T-paths, i.e.,  $\mathbb{P} \in \mathcal{G}^P$ , and the newly generated V-paths, and where weight function  $W^+$  takes as input an edge and a T-path or a V-path and returns the total cost distribution. With V-paths, there is no need to maintain the joint distributions for T-paths and V-paths.

By introducing V-paths, Lemma 4.1 shows that the cost distribution of any path  $P$  can be computed using convolution only, thus offering a theoretical foundation for using stochastic dominance based pruning in PACE. Put differently, by introducing V-paths, Eq. 1 is no longer needed, and we have turned PACE into EDGE, where stochastic dominance-based pruning guarantees correctness.

**LEMMA 4.1.** *Given any path  $P$  in the updated PACE graph  $\mathcal{G}^{P^+}$ , the distribution of  $P$  is computed by using convolution of the weights of edges, T-paths, and V-paths maintained in  $\mathcal{G}^{P^+}$ .*

**PROOF.** In the original PACE model, the distribution  $D_J(P)$  of path  $P$  is computed using Eq. 1 on the coarsest T-path sequence  $CPS(P) = (p_1, p_2, \dots, p_m)$ , where each  $p_i$  is an edge or a T-path that maintains a joint distribution. If adjacent  $p_i$  and  $p_{i+1}$ , where  $i \in [1, m-1]$ , do not overlap, we split  $CPS(P)$  such that  $p_i$  and  $p_{i+1}$  go to two different sub-sequences. Finally,  $CPS(P)$  is split into multiple sub-sequences  $(cps_1, cps_2, \dots, cps_n)$ , where each sub-sequence  $cps_k = (p_j, \dots, p_{j+x})$ , with  $k \in [1, n]$ ,  $j \in [1, m]$ , and  $x \in [0, m-1]$ , does not overlap any others. Here, a sub-sequence  $cps_k$  can be an edge, a T-path, or multiple overlapping T-paths. When  $x > 0$ ,  $cps_k$  includes multiple overlapping T-paths. Since we already built V-paths for overlapping T-paths, there must be a V-path that corresponds to  $cps_k$ . Thus,  $cps_k$  is an edge, a T-path, or a V-path.

Since there are no overlaps among different  $cps_k$ , Eq. 1 yields  $D_J(P) = \prod_{k=1}^n W_J(cps_k)$ , meaning that edges, T-paths, and V-paths are independent of each other. Thus, the cost distribution  $D(P)$  derived from  $D_J(P)$  in the PACE model is equivalent to  $\oplus_{k=1}^n W^+(cps_k)$ , involving only the convolution of items in the weight function  $W^+(\cdot)$  in the updated PACE graph with V-paths.  $\square$

## 4.2 Routing Algorithm

The V-paths introduced in the updated PACE graph  $\mathcal{G}^{P^+}$  enable us to use stochastic dominance to prune non-competitive candidate paths as in the EDGE model. However, after adding the V-paths, the degrees of many vertices also increase. This indicates that when exploring an vertex, more candidate paths may be generated and examined. For example, consider the left most vertex in Figure 7. Before introducing the V-paths, its degree is 2 with edge  $e_1$  and T-path  $p_1$ . After introducing the V-paths, its degree becomes 4. Luckily, with the proposed search heuristics, we are able to choose the most promising candidate paths among many candidate paths. Thus, the increased degrees do not adversely affect the query efficiency.

Algorithm 4 shows the final routing algorithm using two proposed speedup techniques—search heuristics and V-paths. Given a PACE graph  $\mathcal{G}^{P^+}$ , for each candidate path  $P_i$  that departs at time  $t$  and connects source  $v_s$  to an intermediate vertex  $v_i$ , we maintain

two attributes. The first,  $P_i.path$ , refers to the underlying path, i.e., a sequence of edges. For example, the *path* attribute of  $\langle p_4, e_4 \rangle$  in Figure 7 is  $\langle e_1, e_2, e_3, e_4 \rangle$  since  $p_4$  is a V-path, which is a sequence of edges in the road network. We maintain this attribute to avoid cycles in candidate paths. The second attribute  $P_i.maxProb$  is the maximum probability of reaching  $v_d$  within cost budget  $B$  when following  $P_i$  to  $v_i$  and then proceeding to  $v_d$ . This probability can be computed using either the proposed binary heuristic or the budget-specific heuristic.

---

### Algorithm 4: V-PathBasedStochasticRouting

---

**Input:** PACE Graph  $\mathcal{G}^{P^+}$ , source  $v_s$ , destination  $v_d$ , departure time  $t$ , budget  $B$ ;  
**Output:** Path  $P^*$ ;  
1 Priority queue:  $Q \leftarrow \emptyset$ ;  
2 **for** each vertex  $v_i$  that is connected from  $v_s$  by an edge, a T-path or a V-path **do**  
3      $P_i.path \leftarrow tracePath(v_s, v_i)$ ;  
4      $D(P_i) \leftarrow W^+(v_s, v_i)$ ;  
5     **if**  $D(P_i).min + v_i.getMin() \leq B$  **then**  
6          $P_i.maxProb \leftarrow maxProb(P_i, B)$ ; /\* Eq. 3 \*/  
7          $Q.add(P_i)$ ;  
8 **while**  $Q \neq \emptyset$  **do**  
9      $\hat{P} \leftarrow Q.peek()$ ;  
10      $v \leftarrow$  last vertex in  $\hat{P}$ ;  
11     **if**  $v = v_d$  **then**  
12          $P^* \leftarrow \hat{P}$ ;  
13         **break**;  
14     **for** each vertex  $u$  that is connected from  $v$  by an edge, a T-path or a V-path and  $u$  has not appeared in  $P.path$  **do**  
15          $c \leftarrow D(P).min + W^+(v, u).min$ ;  
16         **if**  $c + u.getMin() \leq B$  **then**  
17              $P'.path \leftarrow P.path + tracePath(v, u)$ ;  
18              $D(P') \leftarrow D(P) \oplus W(v, u)$ ; /\* Use convolution to compute a distribution. \*/  
19              $P'.maxProb \leftarrow maxProb(P', B)$ ; /\* Eq. 3 \*/  
20             **if**  $Prune(P', Q)$  **then**  
21                  $Q.add(P')$ ;  
22 **return**  $P^*$ ;

---

We use a priority queue to maintain all candidate paths, where the priority is according to the *maxProb* attribute of each candidate path. In each iteration, we explore the path  $P$  with the largest *maxProb* attribute, as it is the most promising candidate path (line 10). If this path  $P$  has already achieved to the destination  $v_d$ , it is the path with the largest probability of arriving within budget  $B$ . This is because all the other candidates in the priority queue cannot have a larger probability as the heuristics are admissible (lines 12–13).

If path  $P$  has not reached the destination, we extend the path with an adjacent edge, T-path, or V-path to get a new candidate path  $P'$ . The distribution of  $P'$  is computed using convolution (line 18). Finally, we check stochastic dominance between  $P'$  and each path  $\hat{P}$  in  $Q$  that also reaches  $u$ . If  $P'$  stochastically dominates  $\hat{P}$ , we can safely prune  $\hat{P}$  from  $Q$ . If  $\hat{P}$  stochastically dominates  $P'$ , we can safely prune  $P'$ , i.e., not add  $P'$  to  $Q$ . If eventually, no path in

$Q$  stochastically dominates  $P'$ , we add  $P'$  to  $Q$  as a new candidate path (lines 20–21).

## 5 EMPIRICAL STUDY

### 5.1 Experimental Setup

**Road Networks and GPS Trajectories:** We use two pairs of a road network and an associated GPS dataset:  $N_1$  and  $D_1$  covering Aalborg, Denmark, and  $N_2$  and  $D_2$  covering Xi'an, China, where  $N_1$  and  $N_2$  are obtained from OpenStreetMap. The sampling rates of the GPS records in  $D_1$  and  $D_2$  are 1 Hz and 0.2 Hz, respectively. We use travel times extracted from the GPS records as travel costs, and we utilize an existing tool [21] to map match  $D_1$  and  $D_2$  to  $N_1$  and  $N_2$ . The mapped trajectories cover 23% and 4% of the edges in  $N_1$  and  $N_2$ . The covered edges, typically main roads, show high uncertainty. The uncovered edges are typically small roads with low uncertainty. For these, we use speed limits to derive deterministic travel times. After detecting and filtering the abnormal data [3, 15, 16], statistics of the Aalborg and Xi'an data are provided elsewhere [8].

We conduct the empirical analysis using five-fold cross validation. **Stochastic Routing Queries:** A stochastic routing query takes three parameters: a source, a destination, and a travel time budget. We select source-destination pairs from the testing set to obtain meaningful source-destination pairs. We categorize the pairs into groups based on their Euclidean distances (km): (0, 5], (5, 10], (10, 25], and (25, 35]. We ensure that each category has at least 90 pairs. Note that 35 km is a long distance in a city. Following existing studies [26], we focus on intra-city routing because cities often have uncertain traffic and have many alternative paths between a source and a destination. For inter-city travel, there is often limited choices, e.g., using vs. not using highways. We leave the support for country-level, inter-city routing as future work.

Next, we generate meaningful travel time budgets for the source-destination pairs. This is important because too small budgets result in many paths having probability 0, while too large budgets result in all paths having probability 1. For each source-destination pair, we run Dijkstra's algorithm based on a road network with expected travel times as edge weights. This yields a path with the least expected travel time  $\bar{t}$ . Then, we set 5 time budgets that correspond to 50%, 75%, 100%, 125%, and 150% of  $\bar{t}$ , respectively, meaning that the time required to travel between the source-destination pair will not stay beyond the range of  $[50\% \bar{t}, 150\% \bar{t}]$ . This enables us to evaluate a range of meaningful time budgets.

**Routing Algorithms:** We consider a baseline routing algorithm in the PACE model [33] called *T-None* (cf. Section 2.2) that does not use any heuristics to estimate the cost from intermediate vertices to the destination and that does not use any V-paths. Next, we consider routing algorithms using only T-paths with different heuristics:

- *T-B-EU*: Binary heuristic using Euclidean distance divided by the maximum speed limit in the road network to derive  $v_i.getMin()$ ;
- *T-B-E*: Binary heuristic using shortest path trees derived from only edges;
- *T-B-P*: Binary heuristic with shortest path trees derived from both edges and T-paths;
- *T-BS- $\delta$* : Budget-specific heuristics using a specific  $\delta$ .

Finally, we consider V-path based routing algorithms: *V-None* uses no heuristics, and *V-B-P* and *V-BS- $\delta$*  use the above heuristics.

**Parameters:** We vary parameters  $\tau$  among 15, 30, **50**, and 100, and  $\delta$  among 30, **60**, 120, and 240, where  $\tau$  is the trajectory threshold used when generating T-paths (cf. Section 2.1) and  $\delta$  is the finest budget value used in the budget-specific heuristics (cf. Section 3.3). Default values are shown in bold. We conduct sensitivity analyses on the two parameters to identify the most appropriate values.

**Evaluation Metrics and Settings:** We evaluate the runtime and space overhead needed for maintaining the different heuristics.

*Comparison with the EDGE model:* We do not compare the paths returned by the PACE models vs. the EDGE model, since a previous study describes their differences and shows the benefits of the PACE model over the EDGE model [33]. Thus, we focus on evaluating the routing algorithm with different heuristics in the PACE model.

*Time-dependency:* For each network, we build two PACE models using the trajectories from peak (7:00-8:30 and 16:00-17:30) vs. off-peak (others) hours. Results are mainly reported for data set  $D_2$  at peak hours due to the space limitations. Results for data set  $D_2$  at off-peak hours and for data set  $D_1$  are provided elsewhere [8].

**Implementation Details:** All experiments are conducted in Python 3.7.3, and executed on a server with a 64-core AMD Opteron 2.24 GHZ CPU and 528 GB main memory under Ubuntu 16.

### 5.2 Instantiating T-Paths and V-Paths

**Instantiating T-Paths:** If a path is traversed by more than  $\tau$  trajectories, we instantiate a T-path for it. Figure 8(a) shows the numbers of T-Paths on  $D_1$  and  $D_2$ , respectively, when varying  $\tau$ . We group T-paths according to their cardinalities, i.e., the numbers of edges they cover. When the cardinality is 1, a T-path is just one edge. Intuitively, a larger  $\tau$  requires that more trajectories occurred on the T-paths. Thus a larger  $\tau$  yields fewer T-paths. However, large  $\tau$  also yields more accurate travel time distributions for the T-paths.

Next, we evaluate the accuracy of estimating the path cost distributions using the T-paths with different  $\tau$  values. To do so, we traverse each path from the testing set (not used for instantiating T-paths) to instantiate a travel time distribution, i.e., the ground truth distribution. We then use the T-path weights obtained in the training set to estimate the distribution of each path in the testing set using Eq. 1. We use KL-divergence to quantify the distances between the ground truth and the estimated distributions. A smaller KL-divergence value indicates that the estimated distributions are closer to the ground truth.

Based on the five-fold cross validation, we report the 95% confidence interval for the KL-divergence values for different  $\tau$  values in Figure 8(b). When  $\tau$  increases from 15 to 50, the KL-divergence values decrease, suggesting that a large  $\tau$  indeed yields T-paths with more accurate cost distributions. However, when  $\tau = 100$ , the KL-divergence values start increasing, and the 95% confidence interval grows. This is because too few T-paths are instantiated, which in turn affects the accuracy of cost distributions. We choose  $\tau = 50$  as the default value because it provides the most accurate results with substantial amounts of T-paths.

**Instantiating V-Paths:** The numbers of V-paths for varying  $\tau$  values are shown in Figure 8(c). Smaller  $\tau$  values result in more T-paths, which also lead to more V-paths. The cardinalities of V-paths are much higher than those of T-paths because V-paths are obtained by merging T-paths. Next, we report the average and

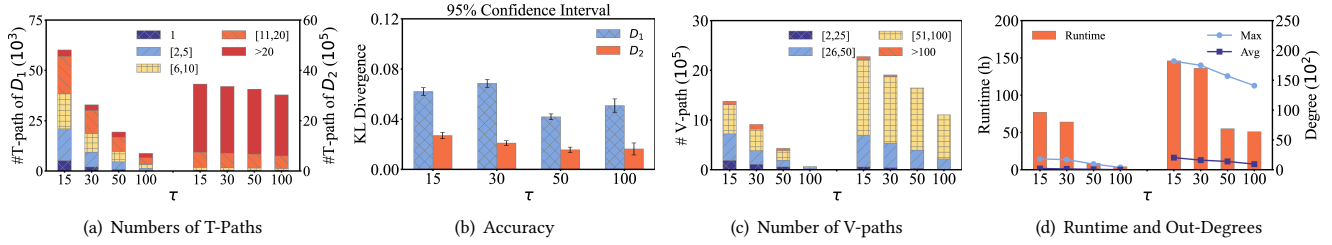


Figure 8: Effect of Varying  $\tau$ : In (a), (c), and (d), the First Four and last Four  $\tau$  Values Correspond to  $D_1$  and  $D_2$ , Respectively

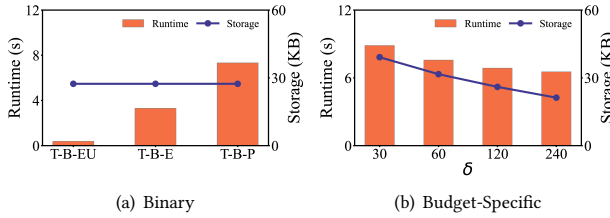


Figure 9: Building Offline Heuristics for Peak Hours,  $D_2$

maximum out-degrees of vertices after introducing the V-paths, in Figure 8(d). The out-degrees are often large because a vertex can now be connected to edges, T-paths, and V-paths. We show that the increased out-degrees do not adversely affect the routing efficiency in Section 5.4, due to the proposed search heuristics. Figure 8(d) also reports the runtime for generating V-paths. This procedure is conducted offline. When using the default of  $\tau = 50$ , it takes around 8.5 and 55 hours for  $D_1$  and  $D_2$ , which is acceptable. Thus, we trade affordable pre-computation time for interactive response times, i.e., at sub 0.1 second level, to be seen next.

### 5.3 Search Heuristics with Only T-Paths

We study both search heuristics when using only T-paths.

**Binary Heuristics:** We consider three variations of binary heuristics: T-B-EU, T-B-E, and T-B-P, as described in Section 5.1. We first study the pre-processing step. Specifically, we study the runtime of computing the  $v.getMin()$  function and the storage needed for the results in data set  $D_2$ , as shown in Figure 9(a).

Recall that the binary heuristics are destination-specific. To support routing queries with arbitrary destinations, we need to maintain a  $v.getMin()$  function for each destination vertex. Figure 9(a) shows the average runtime of  $v.getMin()$  and the storage needed for one destination. T-B-EU is the fastest, as it computes the Euclidean distance between an intermediate vertex and the destination and divides by the maximum speed limit. T-B-E and T-B-P take longer time as they both need to generate shortest path trees. T-B-P takes the longest time as it also involves dominance checking when taking into account the accurate costs maintained in T-paths. Note that T-B-P takes less than 4.0 seconds. We argue that this is reasonable because it is an offline computation and the  $v.getMin()$  function can be computed in parallel for different destinations.

The storage needed for different variations is the same—we maintain a single integer value  $v.getMin()$ , regardless of the variation.

Maintaining  $v.getMin()$  for all destinations takes 0.92 GB, which can easily fit into main memory. More details for building and maintaining the binary heuristics are available elsewhere [8].

We next investigate the runtime when using binary heuristics vs. T-None that uses no heuristics in Figure 10(a) and (b). We categorize the routing queries based on their source-destination distances (cf. Figure 10(a)) and their time budgets (cf. Figure 10(b)).

The longer the distances, the more time it takes to route. Larger travel time budgets also incur longer runtime. T-B-EU takes longer, i.e., sometimes over 0.5 second, than T-B-E and T-B-P at sub 0.1 second, suggesting that the shortest path tree based binary heuristics used by T-B-E and T-B-P are more accurate and are effective at pruning unpromising candidate paths. Further, T-B-P offers the best efficiency, indicating that shortest path tree derived from both edges and T-paths is the most accurate and effective at pruning.

**Budget-specific Heuristics:** We consider the largest budget of 5,000 seconds for both data sets and vary  $\delta$  among 30, 60, and 120, and 240. We argue that 5,000 seconds is a very large time budget when traveling up to 35 km, i.e., the longest distance considered in the routing queries.

We first consider the offline phase of building the budget-specific heuristics, i.e., the heuristics tables. These tables are destination-specific. Figure 9(b) reports the average pre-computation time of constructing a heuristic table and its average size. The results indicate that smaller  $\delta$  values result in larger heuristics tables with more columns and that longer pre-computation times are needed to obtain the heuristics tables.

Next, to investigate how the budget-specific heuristics help stochastic routing, we categorize routing queries w.r.t. both distances and budge values. Figure 10(c) and (d) show that queries with longer distances and travel time budgets incur longer online runtime. Smaller  $\delta$  values produce heuristic tables at finer granularities, thus yielding more effective pruning and lower online runtimes.

We notice that, for  $D_2$  at peak hours, the online runtimes for  $\delta = 30$  and  $\delta = 60$  are similar, but they require 289.2 hours and 247.1 hours (without using parallelization) to generate and occupy 4.38 GB and 3.54 GB to store heuristic tables for all destinations, respectively. Details are available elsewhere [8]. As  $\delta = 60$  requires less pre-computation time and space, we choose it as the default value and conclude that it is practical to trade pre-computation time and storage for interactive response times.

To compare the binary heuristics vs. the budget-specific heuristics (using the default  $\delta$  value), we include T-BS-60 in Figure 10(a)

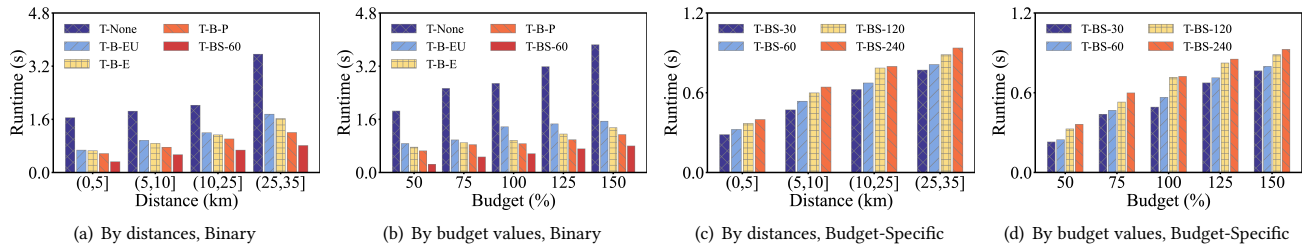


Figure 10: Stochastic Routing with Binary and Budget-specific Heuristics at Peak Hours,  $D_2$

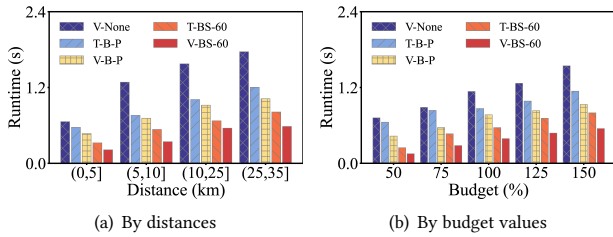


Figure 11: V-Path based Stochastic Routing at Peak Hours,  $D_2$

and (b). We observe that the budget-specific heuristics significantly improve the runtime compared to the binary heuristics.

#### 5.4 V-Path Based Routing

Figure 11 shows results for V-Path based routing, which in general are much faster than routing with only T-paths. In this set of experiments, we only use default  $\delta = 60$ . V-BS-60 achieves the least routing runtime, which further improves over T-BS-60, suggesting that the V-path based routing offers the best efficiency, because using long V-paths often makes it possible to go to somewhere close to the destination very quickly. When combined with stochastic dominance based pruning, the efficiency is further improved.

## 6 RELATED WORK

**Modeling Travel Costs in Road Networks:** Most studies model travel costs in the edge-centric model, meaning that weights are assigned to edges. First, some models assign deterministic weights to edges. An edge weight can be a single deterministic cost representing, e.g., average travel-time [17, 36, 42], or it can be several deterministic costs representing different travel costs such as average travel-time and fuel consumption [9, 10]. Second, some models assign uncertain weights to edges. Most such studies assume that the weights on different edges are independent [24, 32, 35]. Some studies consider the dependency between two edges, but disregards dependencies among multiple edges [13, 26, 34], including two recent studies that employ machine learning models to infer dependencies between two edges where trajectories are unavailable [26, 34]. Path representation learning produces effective path representations to enable accurate travel cost estimation [37–41]. However, such methods are difficult to be integrated with existing routing algorithms. The path-centric model [4, 33] aims at fully

capture the cost dependencies among multiple edges along paths by maintaining joint distributions for the paths when sufficient trajectories are available. The path-centric model offers better accuracy when estimating cost distributions of paths over the edge-centric model [4, 33]. Thus, we base our work on the path-centric model.

**Stochastic Routing:** Stochastic routing is to determine the optimal routes for vehicles in a network under uncertain conditions [18, 19, 25, 28, 31]. Extensive research has been performed on stochastic routing under the edge-centric model [2, 11, 14, 20, 22, 24, 27, 29, 32, 35]. Efficient stochastic routing is often based on stochastic dominance based pruning, based on the assumption that edge weights are independent [12, 24, 32]. In addition, efficient convolution computation techniques have been proposed and integrated into routing algorithms to improve efficiency [5, 22]. Well-known speed-up techniques for classic shortest path finding, e.g., partition-based reach, arc-flags, arc-potentials, and contraction hierarchies, have also been extended to support uncertain weights in the edge-centric model [27, 30]. However, these speed-up techniques rely on the independence assumption that does not hold in the path-centric model, and thus they cannot be applied readily. For the path-centric model, stochastic routing algorithms suffer from low efficiency [1, 33], where we propose search heuristics and virtual-path based routing to improve routing efficiency.

## 7 CONCLUSION

We study stochastic routing under the path-centric model that can capture accurate travel cost distributions. Instead of providing stochastic routing services using edge devices, we target cloud-based stochastic routing services to enable applications such as coordinated fleet transportation. In this setting, sufficient storage is accessible. Thus, we trade increased off-line storage for efficient on-line routing services. Specifically, we propose binary heuristics and a budget-specific heuristic, which help us explore more promising candidate paths. Then, we introduce virtual paths to make it possible to use stochastic dominance based pruning in the path-centric model. Experimental results suggest that the proposed methods enable efficient stochastic routing under the path-centric model.

## ACKNOWLEDGMENTS

This work was partially supported by National Natural Science Foundation of China (62372179) and Independent Research Fund Denmark under agreement 8048-00038B. Bin Yang is the corresponding author of the work.

## REFERENCES

- [1] Georgi Andonov and Bin Yang. 2018. Stochastic Shortest Path Finding in Path-Centric Uncertain Road Networks. In *MDM*. 40–45.
- [2] Russell Bent and Pascal Van Hentenryck. 2007. Waiting and Relocation Strategies in Online Stochastic Vehicle Routing. In *IJCAI*. 1816–1821.
- [3] David Campos, Tung Kieu, Chenjuan Guo, Feiteng Huang, Kai Zheng, Bin Yang, and Christian S. Jensen. 2022. Unsupervised Time Series Outlier Detection with Diversity-Driven Convolutional Ensembles. *Proc. VLDB Endow.* 15, 3 (2022), 611–623.
- [4] Jian Dai, Bin Yang, Chenjuan Guo, Christian S. Jensen, and Jilin Hu. 2017. Path Cost Distribution Estimation Using Trajectory Data. *PVLDB* 10, 3 (2017).
- [5] Brian C Dean. 2010. Speeding up Stochastic Dynamic Programming with Zero-delay Convolution. *Algorithmic Oper. Res.* 5, 2 (2010), 96–104.
- [6] Michael T Goodrich and Roberto Tamassia. 2006. *Algorithm Design: Foundation, Analysis and Internet Examples*. John Wiley & Sons.
- [7] Chenjuan Guo, Christian S. Jensen, and Bin Yang. 2014. Towards Total Traffic Awareness. *SIGMOD Record* 43, 3 (2014), 18–23.
- [8] Chenjuan Guo, Ronghui Xu, Bin Yang, Ye Yuan, Tung Kieu, Yan Zhao, and Christian S. Jensen. 2024. Efficient Stochastic Routing in Path-Centric Uncertain Road Networks – Extended Version. arXiv:2407.06881 [cs.DS] <https://arxiv.org/abs/2407.06881>
- [9] Chenjuan Guo, Bin Yang, Ove Andersen, Christian S. Jensen, and Kristian Torp. 2015. EcoMark 2.0: Empowering Eco-routing with Vehicular Environmental Models and Actual Vehicle Fuel Consumption Data. *Geoinformatica* 19, 3 (2015), 567–599.
- [10] Chenjuan Guo, Bin Yang, Ove Andersen, Christian S. Jensen, and Kristian Torp. 2015. EcoSky: Reducing vehicular environmental impact through eco-routing. In *ICDE*. 1412–1415.
- [11] Chenjuan Guo, Bin Yang, Jilin Hu, Christian S. Jensen, and Lu Chen. 2020. Context-aware, preference-based vehicle routing. *VLDB J.* 29, 5 (2020), 1149–1170.
- [12] Jilin Hu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2018. Risk-aware Path Selection with Time-varying, Uncertain Travel Costs: A Time Series Approach. *VLDB J.* 27, 2 (2018), 179–200.
- [13] Ming Hua and Jian Pei. 2010. Probabilistic Path Queries in Road Networks: Traffic Uncertainty Aware Path Selection. In *EDBT*. 347–358.
- [14] Ruoming Jin, Lin Liu, Bolin Ding, and Haixun Wang. 2011. Distance-Constraint Reachability Computation in Uncertain Graphs. *PVLDB* 4, 9 (2011), 551–562.
- [15] Tung Kieu, Bin Yang, Chenjuan Guo, Razvan-Gabriel Cirstea, Yan Zhao, Yale Song, and Christian S. Jensen. 2022. Anomaly Detection in Time Series with Robust Variational Quasi-Recurrent Autoencoders. In *ICDE*. 1342–1354.
- [16] Tung Kieu, Bin Yang, Chenjuan Guo, Christian S. Jensen, Yan Zhao, Feiteng Huang, and Kai Zheng. 2022. Robust and Explainable Autoencoders for Unsupervised Time Series Outlier Detection. In *ICDE*. 3038–3050.
- [17] Julia Letchner, John Krumm, and Eric Horvitz. 2006. Trip Router with Individualized Preferences (TRIP): Incorporating Personalization into Route Planning. In *AAAI*. 1795–1800.
- [18] Bo Lin, Bissan Ghaddar, and Jatin Nathwani. 2022. Deep Reinforcement Learning for the Electric Vehicle Routing Problem With Time Windows. *TITS* 23, 8 (2022), 11528–11538.
- [19] Ji Lu, Yuning Chen, Jin-Kao Hao, and Renjie He. 2020. The Time-dependent Electric Vehicle Routing Problem: Model and solution. *Expert Syst. Appl.* 161 (2020), 113593.
- [20] Yu Ma, Bin Yang, and Christian S. Jensen. 2014. Enabling Time-Dependent Uncertain Eco-Weights For Road Networks. In *GeoRich@SIGMOD*. 1–6.
- [21] Paul Newson and John Krumm. 2009. Hidden Markov Map Matching through Noise and Sparseness. In *SIGSPATIAL*. 336–343.
- [22] Yu Nie and Yueyue Fan. 2006. Arriving-on-time Problem: Discrete Algorithm that Ensures Convergence. *Transp. Res. Rec.* 1964, 1 (2006), 193–200.
- [23] Mehrdad Niknami and Samitha Samaranyake. 2016. Tractable Pathfinding for the Stochastic On-time Arrival Problem. In *SEA*. Springer, 231–245.
- [24] Evdokia Nikolova, Matthew Brand, and David R. Karger. 2006. Optimal Route Planning under Uncertainty. In *ICAPS*. 131–141.
- [25] Weixu Pan and Shiqiang Liu. 2023. Deep Reinforcement Learning for the Dynamic and Uncertain Vehicle Routing Problem. *Applied Intelligence* 53 (2023), 405–422.
- [26] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. 2020. Anytime Stochastic Routing with Hybrid Learning. *PVLDB* 13, 9 (2020), 1555–1567.
- [27] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. 2020. Fast Stochastic Routing under Time-varying Uncertainty. *VLDB J.* 29, 4 (2020), 819–839.
- [28] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. 2020. A Hybrid Learning Approach to Stochastic Routing. In *ICDE*. 1910–1913.
- [29] Simon Aagaard Pedersen, Bin Yang, Christian S. Jensen, and Jesper Møller. 2023. Stochastic Routing with Arrival Windows. *ACM Trans. Spatial Algorithms Syst.* 9, 4 (2023), 1–48.
- [30] Guillaume Sabran, Samitha Samaranyake, and Alexandre M. Bayen. 2014. Pre-computation Techniques for the Stochastic on Time Arrival Problem. In *ALENEX 2014*. 138–146.
- [31] Yong Wang, Guoliang Li, and Nan Tang. 2019. Querying Shortest Paths on Time Dependent Road Networks. *PVLDB* 12, 11 (2019), 1249–1261.
- [32] Michael P. Wellman, Matthew Ford, and Kenneth Larson. 1995. Path Planning under Time-Dependent Uncertainty. In *UAI1995*. 532–539.
- [33] Bin Yang, Jian Dai, Chenjuan Guo, Christian S. Jensen, and Jilin Hu. 2018. PACE: A Path-Centric Paradigm for Stochastic Path Finding. *VLDB J.* 27, 2 (2018), 153–178.
- [34] Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2013. Travel Cost Inference from Sparse, Spatio-Temporally Correlated Time Series Using Markov Models. *PVLDB* 6, 9 (2013), 769–780.
- [35] Bin Yang, Chenjuan Guo, Christian S. Jensen, Manohar Kaul, and Shuo Shang. 2014. Stochastic Skyline Route Planning under Time-varying Uncertainty. In *ICDE*. 136–147.
- [36] Bin Yang, Manohar Kaul, and Christian S. Jensen. 2014. Using Incomplete Information for Complete Weight Annotation of Road Networks. *IEEE Trans. Knowl. Data Eng.* 26, 5 (2014), 1267–1279.
- [37] Sean Bin Yang, Chenjuan Guo, Jilin Hu, Jian Tang, and Bin Yang. 2021. Unsupervised Path Representation Learning with Curriculum Negative Sampling. In *IJCAI*. 3286–3292.
- [38] Sean Bin Yang, Chenjuan Guo, Jilin Hu, Bin Yang, Jian Tang, and Christian S. Jensen. 2022. Weakly-supervised Temporal Path Representation Learning with Contrastive Curriculum Learning. In *ICDE*. 2873–2885.
- [39] Sean Bin Yang, Chenjuan Guo, and Bin Yang. 2022. Context-Aware Path Ranking in Road Networks. *IEEE Trans. Knowl. Data Eng.* 34, 7 (2022), 3153–3168.
- [40] Sean Bin Yang, Jilin Hu, Chenjuan Guo, Bin Yang, and Christian S. Jensen. 2023. LightPath: Lightweight and Scalable Path Representation Learning. In *KDD*. 2999–3010.
- [41] Sean Bin Yang and Bin Yang. 2020. Learning to Rank Paths in Spatial Networks. In *ICDE*. 2006–2009.
- [42] Jiangchuan Zheng and Lionel M. Ni. 2013. Time-Dependent Trajectory Regression on Road Networks via Multi-Task Learning. In *AAAI*. 1048–1055.