Anytime Stochastic Routing with Hybrid Learning

Simon Aagaard Pedersen Bin Yang ✉ Christian S. Jensen
Department of Computer Science, Aalborg University, Denmark
{sape, byang, csj}@cs.aau.dk

ABSTRACT

Increasingly massive volumes of vehicle trajectory data hold the potential to enable higher-resolution traffic services than hitherto possible. We use trajectory data to create a high-resolution, uncertain road-network graph, where edges are associated with travel-time distributions. In this setting, we study probabilistic budget routing that aims to find the path with the highest probability of arriving at a destination within a given time budget. A key challenge is to compute accurately and efficiently the travel-time distribution of a path from the travel-time distributions of the edges in the path. Existing solutions that rely on convolution assume independence among the distributions to be convolved, but as distributions are often dependent, the result distributions exhibit poor accuracy. We propose a hybrid approach that combines convolution with estimation based on machine learning to account for dependencies among distributions in order to improve accuracy. Since the hybrid approach cannot rely on the independence assumption that enables effective pruning during routing, naive use of the hybrid approach is costly. To address the resulting efficiency challenge, we propose an anytime routing algorithm that is able to return a "good enough" path at any time and that eventually computes a high-quality path. Empirical studies involving a substantial real-world trajectory set offer insight into the design properties of the proposed solution, indicating that it is practical in real-world settings.

PVLDB Reference Format:
Simon Aagaard Pedersen, Bin Yang, Christian S. Jensen. Anytime Stochastic Routing with Hybrid Learning. *PVLDB*, 13(9): 1555–1567, 2020.
DOI: https://doi.org/10.14778/3397230.3397248

1. INTRODUCTION

Users who travel in road networks will generally benefit from high-resolution routing, where travel-time uncertainty is captured accurately [1, 2]. For example, when a vehicle needs to arrive at a destination by a deadline, having accurate travel-time distributions of candidate paths enables choosing the most reliable path. In the example in Table 1, if the deadline is within 60 minutes, path P_1 is better than path P_2, since P_1 offers a 0.8 probability of arriving within 60 minutes, which exceeds P_2's probability of 0.7. If using average travel times, the vehicle will choose P_2 that has average travel time 57, while P_1's average travel time is 60. Thus, the vehicle has a higher risk of arriving too late.

Table 1: Travel-time Distributions of Two Paths P_1 and P_2

Travel time (minutes)	50	60	70
P_1	0.2	0.6	0.2
P_2	0.6	0.1	0.3

Trajectories are increasingly becoming available that capture the movements of vehicles [3]. This provides a solid foundation for high-resolution travel-time uncertainty modeling [4]. We model a road network as a graph, where vertices represent road intersections and edges represent road segments. Then, we split trajectories into pieces that fit the underlying edges and assign uncertain weights in the form of travel-time distributions to the edges using assigned trajectory pieces [5]. The travel-time distributions of edges are often assumed to be independent of each other [6], and the travel-time distribution of a path is computed by applying convolution to the travel-time distributions of the edges in the path.

However, the travel times on edges are often dependent due to traffic lights and turns, etc. Therefore, the independence assumption often causes inaccurate results. Consider an example path $P = \langle e_1, e_2 \rangle$ and assume that a set of trajectories traverse the path. Half of these traverse e_1 in 20 seconds and e_2 in 30 seconds, yielding a total travel time of 50 seconds. The remaining half traverse e_1 in 30 seconds and e_2 in 40 seconds, yielding a total travel time of 70 seconds. This suggests travel time dependence where a driver either traverses both edges quickly or slowly, and it is unlikely that a driver traverses one edge quickly but the other slowly.

In the example, we would split the trajectories into pieces that fit edges e_1 and e_2 such that we obtain distributions D_1 and D_2 for the two edges, as shown in Table 2. The convolution of D_1 and D_2, shown in Table 3, does not reflect the same reality as do the original trajectories (see Table 4). Rather, we now have a large probability of traversing the two edges in 60 seconds, which no trajectories support.

To better capture travel-time dependence, we propose a hybrid learning approach. First, we train a regression model, specifically

Table 2: Distributions D_1 and D_2 for Edges e_1 and e_2

D_1		D_2	
Travel Time	Probability	Travel Time	Probability
20	0.5	30	0.5
30	0.5	40	0.5

Proceedings of the VLDB Endowment, Vol. 13, No. 9
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3397230.3397248

Table 3: Convolution Result

Travel Time	Probability
50	0.25
60	0.50
70	0.25

Table 4: Ground Truth D_P

Travel Time	Probability
50	0.50
70	0.50

a neural network, that is able to take into account two edges' distributions and road condition features that describe the relationships between the two edges, e.g., whether a traffic light is in-between them and the angle between them, to estimate the travel-time distribution of the path that consists of the two edges. In our example, when training the regression model, the inputs are D_1, D_2, and road condition features. The ground truth label for the input is distribution D_P, shown in Table 4.

Second, because convolution is likely a good choice, especially when the dependence between two edges is weak, we train a binary classifier to decide whether or not to use convolution or the trained regression model. This results in a novel hybrid learning approach that yields better accuracy compared to using only convolution or only regression.

In addition, the hybrid learning approach satisfies a so-called "incremental property," which is highly desired in routing algorithms. Specifically, routing algorithms often employ a "path+another edge" pattern to extend an existing path P by an edge e to obtain a new path P'. The hybrid learning is able to use the distribution of P, which is already computed, and the distribution of edge e, to incrementally compute the distribution of the new path P', rather than computing the distribution of P' from scratch, which may take a much longer time. To the best of our knowledge, the hybrid learning approach is the first machine learning based approach that satisfies this important property.

In a stochastic setting, the notion of "shortest" path needs to be redefined because paths have distributions, not deterministic values. We consider a classic stochastic routing problem, *probabilistic budget routing*, where, given a source, a destination, and a time budget, the goal is to identify the path with the largest probability of arriving the destination within the time budget. Existing routing algorithms rely on assumptions that do not hold in the hybrid learning approach, thus calling for new routing algorithms. Specifically, existing routing algorithms rely on stochastic dominance to compare distributions of paths. Let P_1 and P_2 have distributions D_1 and D_2, respectively. If distribution D_1 stochastically dominates distribution D_2, then D_1 is considered to be "smaller" than D_2, and then we can prune P_2 as in the deterministic case. The reason is that when extending P_1 and P_2 by an edge e, the distribution $D_1 \oplus D_e$ of path $P_1' = \langle P_1, e\rangle$ dominates the distribution $D_2 \oplus D_e$ of path $P_2' = \langle P_2, e\rangle$, where $\oplus$ denotes convolution. However, when using a regression model $RM(\cdot,\cdot)$, we cannot guarantee that $RM(D_1, D_e)$ dominates $RM(D_2, D_e)$ when D_1 dominates D_2. This resulting inability to prune based on stochastic dominance when using regression makes efficient routing more challenging. This calls for new routing algorithms.

To this end, we propose an anytime routing algorithm that employs the hybrid learning approach to estimate path distributions along with additional speed-up techniques to ensure efficiency. Specifically, the algorithm is able to deliver a "good enough" path at any time; and the longer the algorithm runs, the better the result path becomes (i.e., the path offers a higher probability of arriving within the budget). This provides flexibility on how long a user may want to wait. The algorithm includes several speeding up techniques, including a method to estimate best possible distributions for paths, enabling an A*-like heuristic, and the use of so-called pivot paths to enable additional pruning.

We make three specific contributions in this paper. (i) We propose a hybrid learning approach to accurately estimate the travel-time distributions of paths while satisfying the incremental property; (ii) we propose an anytime routing algorithm that employs the hybrid learning approach to support probabilistic budget routing; and (iii) we conduct a comprehensive empirical study to justify our design choices and to offer insight into the proposed methods. A preliminary four-page report on the study appeared elsewhere [7].

The remainder of the paper is organized as follows: Section 2 presents preliminaries. Section 3 presents the hybrid learning approach for path travel cost distribution estimation. Section 4 introduces an anytime routing algorithm that utilizes the hybrid learning. Section 5 presents the empirical study. Finally, Section 6 covers related work, and Section 7 concludes and suggests future work.

2. PRELIMINARIES

Uncertain Road Networks: An uncertain road network is defined as a graph $G = (V, E, \mathcal{C})$, where V is a set of vertices v_i, $E \subseteq V \times V$ is a set of edges $e_i = (v_j, v_k)$, and $\mathcal{C}$: $E \rightarrow D$ is a cost function that maps edges to their cost distributions, e.g., travel-time or fuel consumption distributions. We denote the source and destination vertices of an edge e as $e.s$ and $e.d$, respectively.

We use histograms to represent distributions. A histogram is a collection of $(bucket, probability)$ pairs with each $bucket$ representing a cost range. The probabilities in a histogram sum to 1.0, and the probability of all cost values in a bucket is uniform. To obtain a histogram for an edge, we select a cost lower bound lb and a cost upper bound ub such that cost range $[lb, ub)$ covers all costs from the GPS traversals on the edge. Next, we form a histogram with n buckets, where each bucket has width $\lceil (ub - lb)/n \rceil$. Then, we distribute the edge costs to the buckets and compute the probability of each bucket. We define two histograms to be *homogeneous* if they have identical cost bounds and bucket widths. For instance, Figure 1(a) illustrates two homogeneous histograms with cost bounds $[0, 10)$ and bucket width 2.

Paths and Path Costs: A path $P = \langle e_1, e_2, \ldots, e_n\rangle$ is a sequence of edges where $e_i \in E$. We only consider simple paths, in which all vertices are distinct. We denote $P.d$ as the final vertex covered by path P. A subpath P_S of path P is a contiguous subsequence of the edges in P. A pre-path P^{-i} of path P is a sub-path of P with the last i edges removed. Path expansion occurs when a path P is extended by an edge e to obtain $P' = \langle P, e\rangle$. Consider an example path $P = \langle e_1, e_2, e_3\rangle$. Then, $P_s = \langle e_2, e_3\rangle$ is a sub-path of P, $P^{-1} = \langle e_1, e_2\rangle$, $P^{-2} = \langle e_1\rangle$, and $\langle P, e_4\rangle = \langle e_1, e_2, e_3, e_4\rangle$.

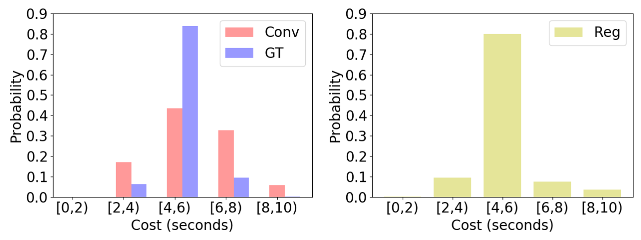
The cost of a path $P = \langle e_1, e_2, \ldots, e_n\rangle$, denoted $P.cost$, is a distribution that is derived from the distributions of the edges in P. When assuming that the distributions are independent, convolution is applied to "sum" the distributions such that $P.cost = \mathcal{C}(e_1) \oplus \mathcal{C}(e_2) \oplus \ldots \oplus \mathcal{C}(e_n)$, where $\oplus$ is the convolution operator. Specifically, the convolution of two independent discrete distributions X and Y can be expressed as $Z(z) = \sum_{x \in X} f_X(x) \cdot f_Y(z - x)$, where f_X and f_Y are the probability mass functions of X and Y. The proposed hybrid learning approach considers potential correlations among the distributions of the edges in path P to compute $P.cost$. The details are covered in Section 3.

Probabilistic Budget Routing (PBR): Given a source s, a destination d, and a time budget t, PBR returns a path P from a path set $\mathbb{P}$ that consists of all paths from s to d, such that P has the largest probability of arriving at d within t. Formally, we have $PBR(s, d, t) = \text{argmax}_{P \in \mathbb{P}}\ Prob(P.cost \leq t)$.

3. HYBRID LEARNING MODEL

3.1 Limitations of Convolution

First, applying convolution to sum two distributions is only accurate if the distributions are independent, which is often not the case in road networks [8]. Second, due to the central limit theorem [9], repeatedly convolving independent distributions eventually yields distributions akin to Gaussian distributions, which often reflect reality poorly in the case of travel-time distributions [6, 8]. Third, the variance of the sum of two independent random variables is the sum of the variances of the two variables. With repeated convolution, we obtain distributions with increasingly large variances, reducing the possibility of any spikes, and instead moving towards distributions that are flatter and more uniform. To exemplify, Figure 1(a) shows the result of convolution and the ground truth using more than 300 GPS traversal records on a path with two edges in Aalborg, Denmark. The convolution result is stretched to be flatter compared to the ground truth that has a spike in bucket $[4, 6)$.



(a) Convolution & Ground Truth (b) Regression

Figure 1: Example based on real-world data [7].

3.2 Learning Path Cost Distributions

We propose to use machine learning to better capture cost distribution dependencies and thus more accurately estimate the cost distributions of paths. We distinguish between cost distribution estimation for short paths and long paths.

3.2.1 Short Paths

We first consider *short paths*, i.e., paths with two edges. We treat path distribution estimation as a regression problem: $\hat{H}_P = F(H_1, H_2, C)$, where H_1 and H_2 are histograms of edges e_1 and e_2, and C represents features that characterize the two edges, e.g., whether they meet at a traffic light or a roundabout. Regression function F estimates the histogram $\hat{H}_P$ of path $P = \langle e_1, e_2 \rangle$. We proceed to elaborate on how to prepare training data and on the design of regression function F.

We employ GPS trajectories for training. We first identify a set of short paths that are traversed frequently by trajectories. For instance, we could select the top-k most traversed short paths or the short paths with more traversals than a threshold.

We then split the selected short paths into disjoint training and testing sets. For each short path $P = \langle e_1, e_2 \rangle$ in the training set, we use all trajectories that traversed e_i to derive histogram H_i for edge e_i. To derive histogram H_P, we only use the smaller set of trajectories that cover the full path $P = \langle e_1, e_2 \rangle$. For instance, suppose that 100 trajectories traversed $P = \langle e_1, e_2 \rangle$, 80 trajectories traversed e_1 but not e_2, and 20 trajectories traversed e_2 but not e_1. Then, we build H_1 (H_2) from the 180 (120) trajectories that traversed e_1 (e_2), and we build H_P from the 100 trajectories that traversed P, i.e., both e_1 and e_2.

Next, we identify features C that characterize the two edges. These include the lengths, road types (e.g., highways vs. residential roads), and speed limits of the two edges; the angle between the edges; and whether there is a traffic light between the edges. The intuition is that the degree of dependence between two edges may be affected by these properties that thus should be considered by the regression model. Some features are discrete and thus one-hot encoded, whereas others are floating values.

We use a classic multilayer perceptron neural network (NN) as the regression model as we expect its ability to capture non-linear relationships among inputs to be essential for capturing distribution correlations. We require that H_1, H_2, and H_P are homogeneous, each having n $(bucket, probability)$ pairs with cost range $[lb, ub)$, and C has c features. Thus, the input layer has $2 \cdot n + c$ neurons. The first $2 \cdot n$ neurons correspond to the probabilities of the n buckets in each of histograms H_1 and H_2, and the last c neurons correspond to the features in C. The hidden layer can have an arbitrary number of neurons. The output layer has n neurons that correspond to the estimated probabilities of the n buckets, i.e., $\hat{H}_P$. The layers are fully connected—each neuron in the input layer is connected to each neuron in the hidden layer with a sigmoid activation function, and each neuron in the hidden layer is connected to each neuron in the output layer with a softmax activation function.

During training, H_P is used as the ground truth, and the squared error between the histograms $\hat{H}_P$ and H_P is used as the loss function. We measure the accuracy of the resulting NN using the test path set. For each testing path $P' = \langle e'_1, e'_2 \rangle$, we derive histograms H'_1, H'_2, and $H_{P'}$. We give H'_1, H'_2, and a feature set that characterizes edges e'_1 and e'_2 to the trained NN, which returns an estimated histogram $\hat{H}_{P'}$ as the cost distribution of path P'. We measure the KL-divergence between the estimated distribution $\hat{H}_{P'}$ and the ground-truth distribution $H_{P'}$. The smaller the KL-divergence is, the more accurate the estimated distribution is.

Figures 1(a–b) show that using regression instead of convolution can yield distributions that are more similar to the ground truth—both the regression-based estimated distribution and the ground truth have a clear spike at bucket $[4, 6)$. Next, we employ KL-divergence to quantify the accuracy between an estimated distribution (using either convolution or regression) vs. the ground truth distribution. A KL-divergence of 0 means that the two distributions are identical, and increasingly dissimilar distributions produce increasingly large KL-divergence scores. In Figure1, the KL-divergence between the ground truth and the distribution derived from regression is 0.06, whereas the KL-divergence between the ground truth and the distribution obtained using convolution is 0.23. This demonstrates that machine learning techniques have the potential to capture distribution dependencies and thus compute more accurate sums of two *dependent* distributions.

Building multiple NNs: Since the input and output histograms of the NN must be homogeneous, we train different NNs for different cost ranges. For example, we train different models to estimate the sum of two short edges in an residential neighborhood vs. the sum of two long edges on a main road. If we were to use a single model, we need to consider histograms that represent the travel times of long edges up to, say, 1,000 seconds. However, the short edges may only have travel times up to 100 seconds, which yields many empty buckets and makes regression more challenging.

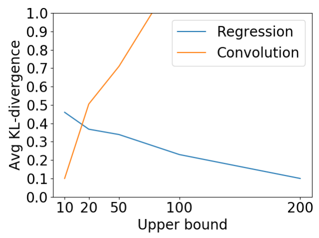
In particular, we consider four filtering parameters to build different NNs: *distance lower and upper bounds* and *cost lower and upper bounds*. We train an NN model based on paths filtered according to the above parameters such that all training paths have a total distance within the distance bounds and the sum of the edges' cost bounds are within the models cost bounds. Because the selected training paths are then similar, their costs can be represented by homogeneous histograms. Having tighter distance and cost bounds results in training paths with higher similarity. However, tighter bounds also yield fewer qualified training paths, and we may eventually not have sufficient similar training paths for training NNs. Thus, we select reasonably tight bounds while making sure that we have sufficient amounts of qualified training paths.

For different NNs, we use a fixed number n of buckets for all histograms and vary the bucket width for each model to represent different cost ranges. Using instead a fixed bucket width and varying number of buckets to represent different cost ranges would require an increasingly large number of buckets, which increases the complexity of the NNs and decreases the accuracy of the regression. Table 5 shows different NNs with different cost and distance bounds, and a fixed number $n = 10$ of buckets.

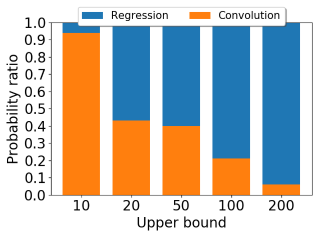
Table 5: Example of Filtering Configurations for Three Models

	Bucket width	Cost bound	Distance bound
M_1	4	(0, 40]	(0, 100]
M_2	20	(0, 200]	(0, 500]
M_3	30	(0, 300]	(0, 500]

Model Selection Given two edges, we select the most appropriate model from a set of available NNs to estimate the cost distribution of the path that consists of the two edges. First, we disregard the NNs whose distance bounds are inappropriate. For example, if an input edge has length 200 m, we disregard M_1 in Table 5 as its distance bound does not cover 200 m. Next, we disregard NNs with inappropriate cost bounds. For example, if the histograms of both input edges range from 0 to 100 seconds, we disregard all NNs whose cost bounds do not cover 200 seconds, e.g., M_1 in Table 5, since the path may take up to $2 \cdot 100 = 200$ seconds. Finally, among all the remaining NNs, we select the NN with the lowest bucket width. In our example, we select M_2 since that has a lower bucket width than M_3.
Model performance Next, we compare NN-based regression with convolution. Recall that the NNs requires all histograms to have a fixed number of buckets, no matter how large the cost ranges they represent. Here, histograms have $n = 10$ buckets. Using trajectories collected from Denmark, we build five different NNs that cover five different cost bounds but within the same distance bound (0, 100] meters. Then, we use each NN to estimate distributions of 500 short paths.



(a) Accuracy



(b) Probability ratio

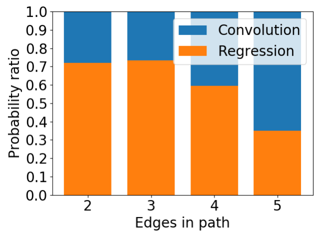
Figure 2: Regression vs. convolution, $n = 10$ buckets.

Figure 2(a) shows the KL-divergence between the estimated histogram (using either regression or convolution) and the ground-truth histogram. Lower KL-divergence values indicate higher accuracy. The accuracy of convolution deteriorates heavily when estimating distributions that represent larger travel-time ranges (i.e., larger upper bounds), whereas the accuracy of regression improves.

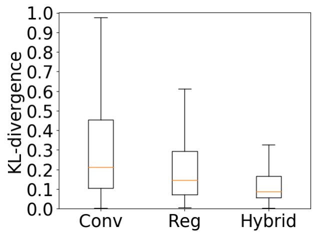
Figure 2(b) shows the probability of regression vs. convolution being more similar to the ground truth. Although the average KL-divergence grows as the cost upper bound increases for convolution, we still find that convolution outperforms regression on a case-by-case basis 40% of the time when the cost upper bound is 50. This suggests that relying purely on either regression or convolution increases result inaccuracy. This calls for a hybrid model that considers both regression and convolution.

3.2.2 *Long Paths*

When performing routing, we need to compute cost distributions for paths that are longer than two edges. To do so, we iteratively apply the NNs. For example, consider a long path $P = \langle e_1, e_2, e_3 \rangle$. We first use an NN to compute the cost of path $P^{-1} = \langle e_1, e_2 \rangle$ using the costs of edges e_1 and e_2. Then we treat path P^{-1} as a virtual edge. This enables us to use an NN to compute the cost of path P using the costs of the virtual edge P^{-1} and edge e_3. When this procedure is applied repeatedly, the virtual edge becomes longer and longer, and the cost ranges of the virtual edge and the last edge become increasingly imbalanced. Since our NN models are not designed for such imbalanced cases, we need to consider the behavior of our NN models when used in this fashion. Figure 3(a) shows the probability of repeated regression vs. repeated convolution being superior for varying path lengths based on 500 samples for each setting using histograms with 10 buckets. Repeatedly applying NN-based regression degrades the accuracy, and regression should not be done more than a few times, at which point convolution becomes preferable. Figure 3(a) shows that regression has a 59% probability of being best for paths consisting of four edges, whereas for paths of five edges, the probability is 34%. This also calls for a hybrid model capable of using both regression and convolution by controlling when regression should be applied.



(a) Probability ratio



(b) Hybrid Model

Figure 3: Estimating the distributions of long paths [7].

3.3 Hybrid Model

To more accurately compute a path's distribution, we create a hybrid model that combines NN-based regression and convolution.

3.3.1 *Boolean Classifier*

We introduce a Boolean classifier to determine whether regression or convolution is to be used in a given context. After training the NN models, we evaluate the accuracy of each NN model on a testing set that consists of short paths. For each test path we also evaluate the accuracy of convolution. Then we know whether regression or convolution is more accurate, and we build a binary classifier using logistic regression based on this information. The classifier is thus associated with a NN model. The classifier accepts a number of features as input and returns two scores that sum to 1.0 and represent the likelihoods of regression vs. convolution being best. We use the same features for both the classifier and the regression model since the underlying problem is the same—to capture the dependence between two edges. Figure 3(b) shows the accuracy of the hybrid approach based on 500 samples. The classifier selects the correct method 73% of the time, yielding an even tighter spread on the KL-divergence of the result. When the classifier is wrong, the average difference in KL-divergence between the incorrectly chosen method and the best method is 0.07. Thus, when the wrong method is selected, the two methods perform very similarly. Using the classifier, we label adjacent edge pairs as being dependent if regression is to be used, and we label them independent if convolution is to be used.

The classifier only determines whether or not two adjacent edges are dependent. This is sufficient given the assumption that pairwise cost dependence between edges is stronger the closer together the edges are: for a path $P = \langle e_1, e_2, e_3\rangle$, where we determine there is no dependence between e_2 and e_3, we also assume that there is no dependence between e_1 and e_3. Inversely, if we determine a cost dependence between e_1 and e_2 and between e_2 and e_3, we also assume a cost dependence between e_1 and e_3, i.e., cost dependence is transitive. We limit the cost dependence reach to a constant k, as we cannot accurately capture the dependence for edges further than k hops away using regression, as shown in Figure 3(a).

Next, we distinguish two different settings for computing the cost distribution of a path using the hybrid model. In the first, a full path is given. In the second, we consider a commonly used path exploration operation in routing, where we do not know the full path, but extend a path by an edge to grow the path.

3.3.2 Computing the Distribution of a Full Path

We consider the case where a full path $P = \langle e_1, e_2, \ldots, e_n\rangle$ is given. Algorithm 1 builds the cost of such a path using the hybrid model. It first determines whether convolution or regression should be used for combining each pair of adjacent edges. Based on the information, we go through each edge and combine the distributions using either convolution or regression. By checking $len = k$ in line 6, we never repeatedly use regression more than $k-1$ times. We evaluate the effect of k in multiple experiments in Section 5.

3.3.3 Incremental Property

In routing, we often need to expand a path by an edge to grow the path. To avoid recomputing the entire cost distribution each time we add an edge to a path, we present an incremental path extension method that caches and reuses different distribution elements of a path. Thus, the hybrid learning approach satisfies an incremental property that is highly preferred in routing. Algorithm 2 details this process. While Algorithm 1 computes the cost distribution of a path given the distributions of all the edges in the path, Algorithm 2 computes the cost distribution of a path $P' = \langle P, e\rangle$ using the distributions of P and e.

In Algorithm 2, we store three different distributions, which we call cost elements: element D_{FC}, representing the cost distribution of a full path; element D_1, representing the cost distribution of the pre-path up to the last independent vertex in the path; and element D_2, representing the cost distribution of the subpath from the last independent vertex in the path to the end vertex.

Whenever a new edge e is to be added to the path, we first determine whether or not the source vertex of e is independent by using a classifier. If it is, we simply convolve the distribution of e with the cost of the path. If not, we combine D_2 with the cost of e using regression and then convolve D_1 and D_2.

Algorithm 1: Path Cost

```
**Input:**
    Path $P = \langle e_1, e_2, \ldots, e_n\rangle$; Integer $k$;
**Output:**
    Cost distribution of path $P$;
 1: For each two consecutive edges in $P$, use classifier to set
    either $v.conv$ or $v.reg$ to true for their shared vertex $v$;
 2: $D \leftarrow$ Distribution of $e_1$;
 3: $S_P \leftarrow$ Set of distributions to be convolved;
 4: $i \leftarrow 2$; $len \leftarrow 1$;
 5: **while** $i \leq |P|$ **do**
 6:     **if** $len = k$ or $e_i.s.conv$ **then**
 7:         Insert $D$ into $S_P$;
 8:         $len \leftarrow 1$;
 9:         $D \leftarrow e_i.cost$;
10:     **else if** $e_i.s.reg$ **then**
11:         $D \leftarrow NN(D, e_i.cost)$;
12:         $len \leftarrow len + 1$;
13:     $i \leftarrow i + 1$;
14: Insert $D$ into $S_P$;
15: $result \leftarrow$ convolve all distributions in $S_P$ into one;
16: return $result$;
```

Algorithm 2: Incrementally Build Path Cost

```
**Input:**
    Path $P = \langle e_1, e_2, \ldots, e_n\rangle$; Edge $e_i$; Integer $k$;
**Output:**
    Path $P' = \langle e_1, e_2, \ldots, e_n, e_i\rangle$ with linked cost elements;
 1: op $\leftarrow$ Use a classifier to decide if convolution or regression
    should be used for calculating the cost of $\langle e_n, e_i\rangle$;
 2: $P' \leftarrow \langle P, e_i\rangle$;
 3: **if** op = convolution **then**
 4:     $P'.D_1 \leftarrow P.D_{FC} \oplus e_i.cost$;
 5:     $P'.D_2 \leftarrow$ empty;
 6:     $P'.D_{FC} \leftarrow P'.D_1$;
 7: **else**
 8:     $P'.D_1 \leftarrow P.D_1$;
 9:     **if** $P.D_2$ is empty **then**
10:         $P'.D_2 \leftarrow e_i.cost$;
11:     **else**
12:         $P'.D_2 \leftarrow NN(P.D_2, e_i.cost)$;
13:     **if** $P.D_1$ is empty **then**
14:         $P'.D_{FC} \leftarrow P'.D_2$
15:     **else**
16:         $P'.D_{FC} \leftarrow P'.D_1 \oplus P'.D_2$;
17: **if** $P'.D_2$ represents the cost of a path of length $k$ **then**
18:     $P'.D_1 \leftarrow P'.D_{FC}$;
19:     $P'.D_2 \leftarrow$ empty;
20: return $P'.D_1$, $P'.D_1$, and $P'.D_{FC}$;
```

To exemplify, consider a path with 7 edges as seen in Figure 4, where the independent vertices are marked with "i", e.g., the vertex between e_3 and e_4.

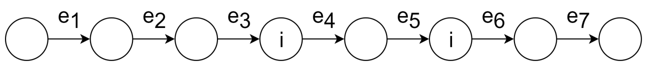


Figure 4: Example graph for path expansion with $k = 3$.

Starting from e_1, we iteratively extend the path with an additional edge. Table 6 shows the different cost elements for each path throughout the expansion. Distribution D_2 always represents the regression result, whereas D_1 represents the established distribution that we only use convolution on. Path P_1 consists of a single edge, and as we have no independent vertex, we store the cost of edge e_1 in D_2. We do not store anything in D_1 before convolution is used for the first time, and the full cost D_{FC} of the path is therefore represented by the distribution given by D_2. Path P_2 has a full cost that is also derived purely from D_2 since there is no independent vertex in P_2. Cost element D_2 now corresponds to the sum of the costs of e_1 and e_2 using regression. We use $k = 3$, meaning that we want to use NN at most two times to estimate the sum of edge distributions before enforcing convolution. Path P_3 consists dependent vertices, and the result of the regression is therefore stored in D_1. Expanding P_3 with e_4 yields path P_4 that has a full cost represented by the result of convolving the contents of D_1 and D_2. Observe that in order to obtain the full cost, we need to perform additional computations, which suggests that storing the total cost as a separate entity is valuable to avoid repetition of the computation. Path P_5 includes e_5 that leads to an independent vertex. As a consequence, we obtain the cost of P_5 by performing regression on the distributions $e_5.cost$ and $P_4.D_2$, and then convolving the result with the distribution $P_4.D_1$. Paths P_6 and P_7 follow the same procedure, representing the total path cost by convolving D_1 and D_2.

Table 6: Cost Elements for Paths; c_n Denotes the Cost of Edge e_n

Path	D_1	D_2	D_{FC}
$P_1 = \langle e_1 \rangle$	empty	c_1	D_2
$P_2 = \langle P_1, e_2 \rangle$	empty	$NN(c_1, c_2)$	D_2
$P_3 = \langle P_2, e_3 \rangle$	$NN(P_2.D_2, c_3)$	empty	D_1
$P_4 = \langle P_3, e_4 \rangle$	$P_3.D_1$	c_4	$D_1 \oplus D_2$
$P_5 = \langle P_4, e_5 \rangle$	$P_4.D_1 \oplus NN(P_4.D_2, c_5))$	empty	D_1
$P_6 = \langle P_5, e_6 \rangle$	$P_5.D_1$	$c_6.c$	$D_1 \oplus D_2$
$P_7 = \langle P_6, e_7 \rangle$	$P_6.D_1$	$NN(c_6, c_7)$	$D_1 \oplus D_2$

Figure 5 shows the accuracy of the hybrid model on long paths. Here, path cardinality denotes the number of vertices in a path. For each path cardinality, we select a minimum of 10 paths with more than 100 trajectories to have sufficient data to construct a ground-truth distribution. Then, we calculate the distribution of each path using Algorithm 2 with varying k and compare with the distribution of the same path derived using only convolution.

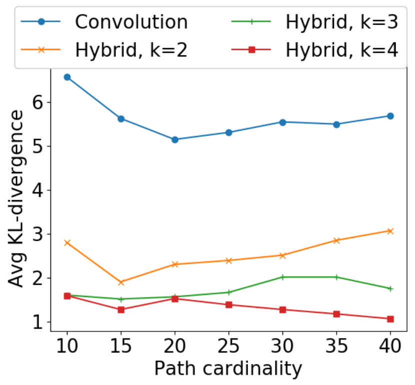


Figure 5: Hybrid model on long paths.

The results show that the hybrid model achieves better accuracy (i.e., lower KL-divergence values). In addition, the hybrid model with a larger k achieves better accuracy. This is due to a larger reliance on the binary classifiers, giving Algorithm 2 more choices of which technique to use for the cost computation. This suggests that using the hybrid model is advantageous.

Discussion: The proposed hybrid learning model distinguishes itself from other machine learning-based path cost estimation methods in two ways. First, existing methods often consider deterministic costs, e.g., average travel times. In contrast, the hybrid model targets cost distributions and explicitly considers distribution dependencies, while existing methods do not touch upon this. Second, existing methods often only consider estimating the cost of a path, i.e., the first setting when a full path is given, and do not satisfy the incremental property. Thus, if using existing methods in a routing algorithm, whenever we extend a path P by e, we need to estimate the cost of path $\langle P, e \rangle$ from scratch without being able to reuse the cost of P. This affects efficiency adversely.

4. ANYTIME HYBRID ROUTING

4.1 Limitations of Existing Routing

Finding the lowest-cost path from a source s to a destination d in a deterministic graph with non-negative edge weights can be solved using Dijkstra's algorithm. Dijkstra's algorithm relies on the *subpath optimality* property. For example, assume that P_1 takes less time than P_2 in Figure 6. As traversal costs are calculated as sums, we can guarantee that $P_1' = \langle P_1, e \rangle$ has a lower cost than $P_2' = \langle P_2, e \rangle$. Thus, at vertex i, we can safely prune P_2 and only consider P_1.

In an uncertain network, the cost of a path is given by a distribution, which makes it more difficult to compare two paths. In this setting, the lowest-cost path can be defined as the path with a stochastically non-dominated distribution among all paths with the same source and destination [1, 2, 6]. Thus, we have to compare distributions to determine dominance relationships.

Given two discrete distributions D_1 and D_2, and their corresponding cumulative distribution functions CDF_{D1} and CDF_{D2}, we say that D_1 dominates D_2 if:

$$\forall x(CDF_{D_1}(x) \geq CDF_{D_2}(x)) \wedge \exists x(CDF_{D_1}(x) > CDF_{D_2}(x)),$$

where x is a travel cost. Thus, the cumulative probability of any cost x in D_1 is never lower than that of D_2, and there is at least one cost x for which D_1 has a strictly larger cumulative probability than has D_2. If P_i's distribution dominates that of P_j, P_i always gives a higher or equal probability of arrival within a time budget no matter what time budget is considered. In other words, given P_i, P_j is not interesting and thus can be pruned.

Based on the notion of stochastic dominance, existing routing algorithms employ a similar strategy to prune uncompetitive paths, as the *subpath optimality* property still holds [2, 10, 11]. For instance, consider Figure 6 where paths P_1 and P_2 reach vertex i, and assume that the distribution of P_1 stochastically dominates that of P_2. In a traditional setting, where we use convolution to compute the cost distributions of paths, we have $P_1' = P_1 \oplus e$ and $P_2' = P_2 \oplus e$, where $\oplus$ denotes convolution, and we use P_i and e to also indicate the distributions of path P_i and edge e. Since P_1 stochastically dominates P_2, it also holds that $P_1 \oplus e$ also dominates $P_2 \oplus e$. Then, at vertex i, P_2 can be pruned.

However, when using machine learning methods to estimate path cost distributions, we may not be able to perform pruning based on stochastic dominance as in the traditional setting. This is because we cannot guarantee that the dominance relationship between two

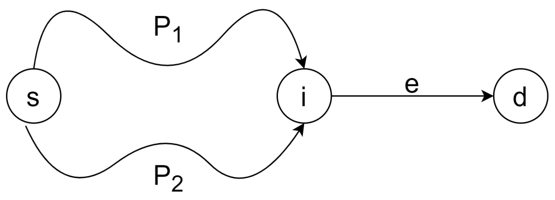


Figure 6: Two paths being extended by the same edge.

paths also holds for the paths when they are extended by the same edge. For instance, in Figure 6 assume that the distribution of P_1 stochastically dominates that of P_2. With hybrid learning, we cannot guarantee that $P'_1 = NN(P_1, e)$ dominates $P'_2 = NN(P_2, e)$, as a neural network is a complex, non-linear function. This breaks the subpath optimality property, and affects pruning.

Thus, when using Dijkstra's algorithm, and replacing convolution with regression, pruning cannot be done on all vertices. As a consequence, we cannot apply Dijkstra's algorithm directly in combination with regression-based cost estimation. However, we can construct a pruning-free, brute-force variant of Dijkstra's algorithm with a priority queue that contains all explored paths organized according to the cost expectation. An example is shown in Algorithm 3. This approach is naturally very inefficient, as the entire search space is explored before a path is returned.

Algorithm 3: Brute-force Search

```
Input:
    Graph $G = (V, E, W)$; Time budget $t$;
    Source $s$ and destination $d$;
Output:
    Path from $s$ to $d$ with max probability of arriving by $t$;
1:  $PQ \leftarrow$ priority queue of paths sorted on expected cost;
2:  Insert all outgoing edges from $s$ into $PQ$;
3:  $result \leftarrow$ set of non-dominated paths to be returned;
4:  while $PQ$ is not empty do
5:      $P \leftarrow$ extract-min from $PQ$
6:      for all $e \in$ outgoing neighbors of $P.d$ do
7:          $P' \leftarrow \langle P, e \rangle$, calculate cost using Algorithm 2;
8:          if $P'.d = d$ then
9:              $result \leftarrow \text{argmax}_{x \in \{P', result\}} prob(x, t)$;
10:         else
11:             if $P'$ is a simple path then
12:                 insert $P'$ into $PQ$;
13: Return $result$;
```

Algorithm 3 returns the best path given budget t between vertices s and d. It iteratively expands the search from s, extending the cheapest path, and it only prunes a path candidate if it no longer is simple or if d is found. The path in $result$ is the best current path found between s and d, and whenever a new path between s and d is found, we compare it to $result$. We use Algorithm 2 in line 7 to build the cost of path P', as it enables the reuse of distributions that are already computed for P.

Algorithm 3 is inefficient mainly because no pruning can be applied at intermediate vertices, and a path can only be returned in the end. This is not acceptable in practice because users would have to wait too long to see a recommended path.

Instead, we propose an anytime routing algorithm. The main consideration is that since we cannot prune paths at all intermediate vertices, the routing may take a long time. Rather than waiting for a long time to return the path with the largest probability of arriving within the budget, we compute a "good enough" path instantaneously and try to efficiently identify better paths (i.e., with higher probabilities). By doing so, we are able to return a path at any time; and as time passes, the algorithm is able to return a path with higher and higher probability.

4.2 Anytime Hybrid Routing

For anytime hybrid routing, we need an additional input x that specifies an acceptable maximum running time, e.g., 3 seconds. The algorithm then stops after x time units. The algorithm uses a so-called pivot path. Specifically, we use simple heuristics to quickly identify a pivot path and then keep improving the pivot path such that its probability of arriving within the time budget increases. Then we can return the pivot path at any time.

Further, the algorithm, detailed in Algorithm 4, employs several pruning techniques to improve performance. The pruning techniques include (a) using an A* inspired optimistic cost of reaching the destination for each vertex, (b) using a pivot path that represents the most promising return candidate at any point during the search in combination with (c) distribution cost shifting that enables comparison between the pivot path and any path that does not yet reach the destination vertex d, and (d) stochastic dominance pruning on all fully independent vertices, i.e., vertices where all combinations of in-edges and out-edges are classified as independent.

We initiate a hybrid search by performing three Dijkstra searches. First, we conduct a one-to-all Dijkstra search from the source vertex on a graph where each edge is associated with the minimum travel cost of its distribution. Then we identify a sub-graph $G' = (V', E')$, where V' represents all reachable vertices from source s within time t and E' includes the edges whose incident vertices are in V'. This sub-graph excludes the vertices and edges that are not reachable from s within time t when always using the most optimistic traversal cost, i.e., always using the minimum travel cost. If V' does not include the destination d, it is impossible to reach d within time budget t.

Second, to enable A*-like search, we need an admissible heuristic for estimating a cost from each vertex to the destination vertex d. An admissible heuristic is one that overestimates the cost of reaching the destination. To this end, we perform a one-to-all Dijkstra search from destination d based on graph G', where each edge is annotated with the minimum cost from the edge's distribution. Then we label each vertex v with $v.min$, i.e., the least required cost of reaching the destination vertex from v. Doing this, we also determine the fastest optimistic path P_a when every edge contributes its minimum cost. We treat P_a as the pivot path for now.

Third, we perform a new Dijkstra search on G', where each edge is annotated with the maximum travel cost, thus identifying the fastest pessimistic path P_b. Then, we use Algorithm 1 to compute the distributions of both paths P_a and P_b. The path with the higher probability of arriving at the destination within time budget t is saved as $pivot$.

The algorithm next explores paths using a priority queue PQ. The queue is sorted in increasing order on (a, b), where a is the optimistic probability of arriving at d by t and b is the expected cost of reaching destination d. The intuition is that we wish to find as many promising candidate paths as possible before termination, and sorting on the A* cost estimate means that paths closer to the destination have a larger priority than paths further away.

Specifically, for a path P from s to vertex v, we shift the cost distribution D_1 of the path to the right by $v.min$ to derive the optimistic distribution of continuing along the path to d, which is used as the A* cost estimation. This ensures that the A* cost estimation is admissible, as it uses the most optimistic cost $v.min$ to destina-

```
Algorithm 4: Anytime Hybrid Search
Input:
    Graph $G = (V, E, W)$; Time budget $t$;
    Source $s$ and destination $d$; Runtime limit $x$;
Output:
    Path with maximum probability of arriving at $d$ by $t$ after
    the algorithm runs $x$ seconds;
1: $t_{start} \leftarrow$ start time;
2: $H_{PS} \leftarrow$ hash map of (vertex, path set);
3: $V' \leftarrow$ all vertices reachable from $s$ within $t$ time using the
   minimum traversal time of each edge;
4: if $V'$ does not contain $d$ then
5:     Return;
6: One-to-all Dijkstra search from $d$ using minimum
   travel costs, annotating all vertices with the
   minimum cost of reaching $d$; and identify the fastest
   optimistic path $P_a$;
7: One-to-all Dijkstra search from $d$ using maximum
   travel costs, annotating all vertices with the
   maximum cost of reaching $d$; and identify the fastest
   pessimistic path $P_b$;
8: Use Algorithm 1 to compute the cost distributions of $P_a$
   and $P_b$;
9: $pivot \leftarrow \text{argmax}_{i \in \{P_a, P_b\}} Prob(i, t)$;
10: if $Prob(pivot, t) = 1.0$ or $P_a.edges = P_b.edges$ then
11:     Return $pivot$;
12: $PQ \leftarrow$ priority queue of paths sorted on (shifted prob
    of reaching $d$ at $t$, expected cost + estimated cost to $d$);
13: Insert all outgoing edges from $s$ into $PQ$;
14: while $PQ$ is not empty do
15:     if Time elapsed since $t_{start}$ is larger than $x$ then
16:         Return $pivot$;
17:     $P \leftarrow PQ.ExtractMin()$;
18:     if $Prob(P.D_1 + P.d.min, t) < Prob(pivot, t)$ then
19:         Return $pivot$;
20:     if $P.d = d$ then
21:         $pivot \leftarrow \text{argmax}_{i \in \{pivot, P\}} Prob(i, t)$;
22:         Continue;
23:     else if $P.d$ is a fully-independent vertex then
24:         Insert $P$ into $H_{PS}[P.d]$ and update such that no
            dominance occurs within $H_{PS}[P.d]$;
25:         Remove all paths $P_m$ from $PQ$ that go through $P.d$,
            where sub-path $P'_m = \langle s, \ldots, P.d \rangle \notin H_{PS}[P.d]$;
26:         if $P \notin H_{PS}[P.d]$ then
27:             Continue;
28:     for all $e \in$ Outgoing edges of $P.d$ where $e.d \in V'$ do
29:         $P_k \leftarrow \langle P, e \rangle$;
30:         Calculate cost of $P_k$ using Algorithm 2;
31:         if $P_k$ is not simple then
32:             Continue; // Paths with loops are never best
33:         $L \leftarrow P_k.D_1$ shifted $e.d.min$ to the right; // Most
            optimistic probability of reaching $d$ within $t$;
34:         if $Prob(L, t) \geq Prob(pivot, t)$ then
35:             Continue;
36:         Insert $P_k$ into $PQ$ with queue cost
            $(Prob(L, t), expectation(P_k) + e.d.min)$;
37: Return $pivot$;
```

tion d that never overestimates the actual cost. For example, shifting the histogram $\{([3,5), 0.3), ([5,7), 0.7)\}$ by $v.min = 6$ to the right yields histogram $\{([9,11), 0.3), ([11,13), 0.7)\}$. We use D_1 and not D_{FC} because doing so may lead to a path expansion with an increased likelihood of arriving by t, whereas using D_1 ensures a probability that is strictly non-increasing during path expansion. This property is important for the statement in line 19, which returns the pivot path if all other paths in the queue have a smaller optimistic probability of arrival by t.

Based on the shifted distribution, we are able to compute b, i.e., the optimistic probability of arriving at d by t.

Recall that the two final searches determine the fastest optimistic and pessimistic paths between s and d in a deterministic setting. The best of these two with respect to time budget t is chosen as a pivot path. Whenever we find a new path reaching the destination, we compare it to the pivot and select the best of the two as the new pivot, in lines 20–22. The pivot path is guaranteed to have a non-zero probability of reaching d within t and is used for comparison with path candidates throughout the algorithm.

The comparison can be performed between the pivot and any path P with an arbitrary end vertex v by shifting P's cost distribution $v.min$ values, where $v.min$ is the minimum traversal time of reaching d from v, which is available via the second Dijkstra search. After the distribution shift, we calculate the probability of arriving by t—if the probability is lower than that of the pivot, we can disregard P as it cannot possibly lead to a path with a higher probability of arriving by t than the pivot path. If the probability is the same or exceeds that of the pivot, we proceed using P.

Finally, we perform stochastic dominance based pruning on all fully independent vertices, in lines 23–27. A vertex is fully independent if the vertex is classified as independent for all combinations of its in-edges and out-edges. Thus, when extending a path that reaches a fully independent vertex v, no matter which edge is followed from v, convolution is always used to compute the cost distribution of the extended path. Thus, it is safe to prune using stochastic dominance.

We use a hash map H_{PS} (line 2) to maintain a path set for each fully independent vertex n containing all non-dominated paths between s and n. Whenever a new path P between s and a fully independent vertex n is discovered, we compare P to all previously discovered non-dominated paths ending at n to determine whether it is necessary to expand P any further. The fully independent vertices are different from the independent vertices used in Algorithm 2. In Algorithm 2, a path is given, so we take into account only the specific edge pair from the path when classifying a vertex as independent or not.

In Algorithm 4, we build path costs in two cases. During routing, we add an edge to an existing path (line 30); and during the initial stage, we build the costs for the fastest optimistic and pessimistic paths (line 8). These two use different inputs. When we expand a path with a single edge, we use Algorithm 2, whereas when we compute the cost of a full path, we use Algorithm 1.

The running time of Algorithm 4 is dominated by the regression and convolution operations. The algorithm can return the pivot path at any time after x time units. The pivot path is improved as the algorithm runs, and the closer the algorithm is to termination, the more likely it is that the pivot path is different from P_a or P_b.

The anytime property breaks the guarantee of returning the path with the largest probability of arriving by t. Instead, Algorithm 4 returns a path that is at least as good as P_a, the best path using deterministic minimum edge costs. In the next section, we assess the quality of the paths returned.

Finally, we note that Algorithm 4 can be used as a routing framework for future stochastic travel cost building models that use machine learning. In order to utilize another cost model, one can change the cost building strategy in lines 8 and 30.

5. EMPIRICAL STUDY

5.1 Experimental Setup

Road network and GPS Data. We use the road networks of Denmark and Chengdu, China, both extracted from OpenStreetMap (`http://www.openstreetmap.org`). The Danish road network consists of 667,950 vertices and 1,647,724 edges, and comes with some 180 million GPS records. We call this data set *DK*. The road network of Chengdu consists of 27,671 vertices and 38,722 edges and comes with some 35 million GPS records. We call this data set *CD*. We map match the GPS data to the underlying road networks.

Uncertain Road network. We instantiate the cost function $\mathcal{C}: E \rightarrow D$ in G as follows: if an edge is covered by GPS records, we derive a travel-time distribution using the GPS records. Otherwise, we derive a travel time tt based on the length and speed limit of the edge. Then, we generate a triangular distribution centered around $tt \cdot 1.2$ with lower bound tt and upper bound $tt \cdot 1.4$. We utilize triangular distributions because they enable explicit cost bounds. The intuition is that drivers may not always drive as fast as the speed limit due to traffic and often spends more time than tt. This approach ensures uncertainty across all edges.

Time Budgets. The time budgets in probabilistic budget routing impact routing efficiency significantly. Selecting a very large budget t enables Algorithm 4 to return one of the two paths found in the deterministic searches, since each path has probability 1.0 of arriving at destination d by t. Conversely, a very small t decreases the search space given by V', which in turn improves efficiency. However, if budget t is too small, no path is able to reach d by t. Thus, we need to select time budgets carefully.

Given a source and destination pair, we run Algorithm 4, but terminate after line 6 to obtain P_a, the fastest optimistic path when all edges are annotated with their minimum costs. Next, we first compute the cost distribution of P_a using the hybrid model and then choose time budgets b_1, b_2, and b_3 such that the probabilities that path P_a has travel time smaller than b_1, b_2, and b_3 are 25%, 50%, and 75%, respectively. This yields meaningful budgets.

PBR Queries. We focus on routing in intra-city settings as the traffic uncertainty in cities is higher and multiple paths often exist between a source and a destination. Inter-city travel paths often use highways, and the choices are limited.. Thus, we generate 100 source-destinations pairs in cities based on three distance (km) categories: $[0, 1)$, $[1, 5)$, and $[5, 10)$. Each pair is associated with three time budgets as described above.

Before performing the routing, we compute the dependence relations in the graph using the binary classifiers described earlier. Edges with no trajectory coverage are assumed to be independent in all associated relations. Roughly 75% of all edge pairs with costs derived from observed trajectories are cost dependent. We learn 10 different *NNs* for summing dependent distributions with different cost and distance bounds (cf. Section 3.2.1).

Baselines. We consider paths returned by four methods. (1) P_∞ is the path returned by Algorithm 4 when it has finished all computations, which is considered as the optimal path for a query. (2) P_x is the path returned by Algorithm 4 after running x seconds, which simulates an anytime routing. Specifically, we consider paths P_1, P_5, and P_{10}. (3) $P_\oplus$ is the path returned by a stochastic baseline method that only uses convolution to compute path cost distributions. This approach assumes all costs are independent [11]. (4) P_0 is the path returned by a deterministic baseline, namely the path identified in line 9 in Algorithm 4.

Implementation. All algorithms are written in Python version 3.7. The experiments are conducted in a single process on a machine running Windows 10 with an 8-core Intel 8900K 4.2 GHz CPU with 32GB DDR4 main memory.

5.2 Experimental Results

5.2.1 Comparisons with Baselines

We compare P_∞ to both deterministic baseline P_0 and stochastic baseline $P_\oplus$. If P_∞ and P_0 are the same in most cases, there is no need to consider stochastic costs, i.e., cost distributions, and thus no need for the hybrid model. Table 7 shows the average percentage that P_∞ is different from P_0. For short queries, we often obtain the same paths, as P_0 has a high likelihood of being the best path because most paths are so short that there is little uncertainty. However, as the query distance increases, P_∞ and P_0 are increasingly dissimilar, on both data sets.

Following the same idea, we examine the percentage of P_1, P_5, and P_{10} being different from P_0. For each query, it is always the case that if $P_x \neq P_0$ then $P_\infty \neq P_0$, and if $P_\infty = P_0$ then $P_x = P_0$. As a consequence, no P_x can have a larger percentage of being different from P_0 than does P_∞. Since all short queries finish within a second, P_1, P_5, and P_{10} are identical to P_∞ thus having the same percentages. However, for longer queries, the uncertainty makes a significant difference.

Table 7: Comparisons with Deterministic Baseline P_0

	DK				*CD*			
	P_∞	P_1	P_5	P_{10}	P_∞	P_1	P_5	P_{10}
$[0, 1)$	13%	13%	13%	13%	24%	24%	24%	24%
$[1, 5)$	53%	51%	53%	53%	65%	61%	63%	63%
$[5, 10)$	60%	54%	59%	60%	90%	66%	71%	74%

We observe similar results on both data sets for the first two distance categories. However, for the longest distance category, we observe a large difference between the two data sets. For *DK*, P_∞ is different from P_0 in 60% of the cases, while for *CD*, the percentage is up to 90%. This suggests that *CD* exhibits large uncertainty and dependence. In addition, for *DK*, P_5 is already very close to P_∞, suggesting that it may be acceptable to have a time limit of 5 seconds for long queries. In contrast, *CD* has a set of queries that do not find a path different from P_0 after 10 seconds, but eventually do. This suggests that the routing algorithm has a large search space due to a high degree of uncertainty.

Next, we examine in detail the cases where P_∞ and P_0 are different. Here, we also include the stochastic baseline $P_\oplus$. If P_∞ and $P_\oplus$ differ in most cases, a strong cost dependence occurs in the search space; otherwise, paths P_∞ and $P_\oplus$ should be identical. This suggests that if there is no strong cost dependence, there is also no need to use hybrid routing.

Given two baseline paths P_0 and $P_\oplus$, we use Jaccard similarity [12] to measure their path similarities against the optimal path P_∞. The Jaccard similarity between paths P and P_∞ is

$$PathSim(P, P_\infty) = \frac{|P \cap P_\infty|}{|P \cup P_\infty|},$$

where each path is represented by its vertices.

Tables 8 and 9 report the average path similarities against P_∞. The similarity between the optimal path P_∞ and the deterministic baseline path P_0 is low. This suggests that stochastic modeling is necessary. Next, stochastic baseline $P_\oplus$ has higher similarities than deterministic baseline P_0 has, which further justifies the stochastic modeling. More importantly, we observe that $P_\oplus$ is still

Table 8: Path Similarity Against P_∞, when $P_\infty \neq P_0$, *DK*

		25%			50 %			75 %		
		k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
$[0,1)$	P_0	0.47	0.40	0.45	0.43	0.37	0.42	0.39	0.34	0.41
	$P_\oplus$	0.65	0.64	0.62	0.67	0.57	0.59	0.63	0.55	0.59
$[1,5)$	P_0	0.47	0.46	0.49	0.47	0.43	0.47	0.46	0.44	0.46
	$P_\oplus$	0.59	0.57	0.55	0.61	0.57	0.57	0.63	0.59	0.58
$[5,10)$	P_0	0.48	0.47	0.45	0.48	0.50	0.48	0.47	0.48	0.48
	$P_\oplus$	0.56	0.52	0.48	0.59	0.56	0.53	0.57	0.57	0.53

Table 9: Path Similarity Against P_∞, when $P_\infty \neq P_0$, *CD*

		25%			50 %			75 %		
		k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
$[0,1)$	P_0	0.31	0.27	0.29	0.33	0.27	0.25	0.30	0.24	0.20
	$P_\oplus$	0.91	0.87	0.86	0.89	0.89	0.86	0.90	0.84	0.85
$[1,5)$	P_0	0.52	0.53	0.50	0.50	0.51	0.50	0.50	0.49	0.51
	$P_\oplus$	0.83	0.81	0.79	0.83	0.84	0.82	0.84	0.87	0.86
$[5,10)$	P_0	0.44	0.43	0.40	0.42	0.40	0.40	0.40	0.41	0.40
	$P_\oplus$	0.80	0.80	0.79	0.81	0.80	0.80	0.80	0.77	0.77

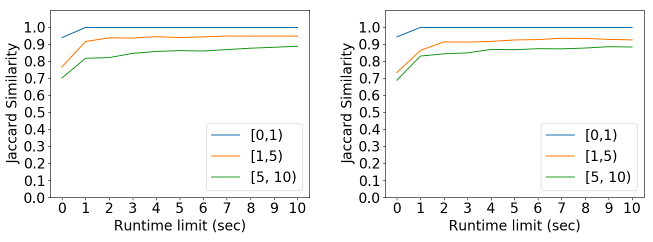
quite different from P_∞. This suggests that considering distribution dependencies makes a difference and is important. Comparing Tables 8 and 9, we find that $P_\oplus$ is much closer to P_∞ for *CD* than for *DK*. This suggests that *CD* exhibits less dependence. Finally, parameter k and different time budgets do not significantly impact the path similarity.

The above experiments indicate that it is essential to consider cost distributions and dependencies among distributions, especially for medium and long queries. Only the hybrid model does this.

5.2.2 *Quality of Anytime Routing*

We proceed to further examine the quality of anytime routing. We observe that the returned paths P_x with different time limits may vary greatly, while the probability of arrival within the time budget t never decreases. Thus, we design two metrics to examine the quality of P_x. The first considers the path similarity between P_x and optimal path P_∞. The second compares the arrival probabilities of P_x and P_∞.

Path similarity: We continue to use Jaccard similarity to measure the similarity between paths P_x and P_∞. We first construct a path similarity profile for Algorithm 4 to show how the path similarity improves as runtime limit x increases. Here, we include all queries, even when $P_0 = P_\infty$, to better capture the path quality a user can expect as a function of x. This experiment is conducted on *DK* with $k = 2$, meaning that we apply regression at most once before switching to convolution. We omit figures for a 50% time budget due to space limitations. Experiments on *CD* give similar results.



(a) Time budget 25%

(b) Time budget 75%

Figure 7: Path similarity profile, *DK*, $k = 2$.

Table 10: Path Similarity Against P_∞ when $P_\infty \neq P_0$, *DK*

		25%			50 %			75 %		
		k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
$[1,5)$	P_1	0.90	0.89	0.88	0.88	0.88	0.88	0.87	0.88	0.87
	P_5	0.95	0.94	0.92	0.95	0.93	0.93	0.93	0.93	0.91
	P_{10}	0.95	0.96	0.93	0.95	0.95	0.94	0.93	0.95	0.92
$[5,10)$	P_1	0.81	0.82	0.78	0.82	0.81	0.79	0.82	0.79	0.78
	P_5	0.85	0.86	0.84	0.88	0.84	0.83	0.86	0.84	0.83
	P_{10}	0.88	0.87	0.84	0.88	0.86	0.85	0.87	0.85	0.85

Table 11: Path Similarity Against P_∞ when $P_\infty \neq P_0$, *CD*

		25%			50 %			75 %		
		k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
$[1,5)$	P_1	0.88	0.83	0.86	0.86	0.80	0.83	0.83	0.84	0.85
	P_5	0.94	0.91	0.90	0.91	0.84	0.88	0.90	0.86	0.88
	P_{10}	0.96	0.92	0.92	0.92	0.86	0.89	0.91	0.87	0.88
$[5,10)$	P_1	0.67	0.70	0.68	0.67	0.70	0.69	0.69	0.69	0.67
	P_5	0.82	0.80	0.76	0.80	0.79	0.74	0.80	0.77	0.74
	P_{10}	0.89	0.87	0.80	0.88	0.86	0.78	0.89	0.85	0.80

Figure 7 shows that the larger the running time limit is, the more similar path P_x tends to be to path P_∞. Path P_0 has the smallest value, indicating that only considering deterministic costs is suboptimal. We see the largest similarity increase when x increases from 0 to 1 or 2. This suggests that there is good potential to use anytime routing.

Next, we examine the path similarity of the paths returned by anytime routing in more detail. We only consider the cases where $P_0 \neq P_\infty$. We disregard distance category $[0,1)$ where P_1 is identical to P_∞ due to the short distance.

Table 10 shows the results on *DK*. Paths P_x and P_∞ in category $[1,5)$ are very similar (above 85%), even when limiting the search to 1 second. This is because the median execution time is just below one second, and $P_1 = P_\infty$ for 75% of the queries. Similarly, using only one second for category $[5,10)$ queries yields the same result as P_∞ in most cases, and all tested time limits yield results very similar to P_∞. Further, using 10 seconds instead of 5 offers little benefit. Also, the path similarities of P_x are higher than those of $P_\oplus$ (cf. the $P_\oplus$ rows in Tables 8 and 9). This suggests that although anytime routing may not return optimal paths, it is able to provide better paths than the stochastic baseline that assumes distribution independence, suggesting that anytime routing is good enough.

We observe similar trends on *CD*—see Table 11. However, we find a larger difference between P_1 and P_{10}, indicating that we need wait longer time to obtain a path that is better than $P_\oplus$.

Examining values for k in Tables 10 and 11 reveals that larger values of k generally results in lower path similarity to P_∞. This is likely due to the increased pruning opportunities when k is low. We can, to a larger extent, utilize stochastic dominance pruning on nodes that several paths pass through when using convolution. This also helps anytime routing to identify the optimal path faster.

Probability similarity: We define a relative arrival probability improvement ratio SR. In particular, given time budget t, we have

$$SR(P_x) = \frac{Prob(P_x, t) - Prob(P_0, t)}{Prob(P_\infty, t) - Prob(P_0, t)}.$$

Here, $Prob(P_\infty, t)$ is the arrival probability within t using optimal path P_∞, which gives an upper bound on the arrival probability. Similarly, $Prob(P_0, t)$ is the arrival probability of using path P_0 that only considers deterministic costs, which serves as the lower

Table 12: Absolute Improvements when $P_x \neq P_0$, *DK*

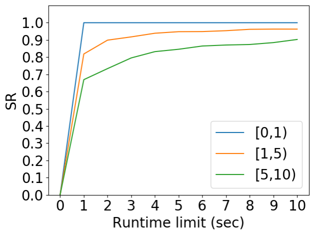
		25%			50 %			75 %		
		k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
$[0,1)$	P_∞	0.38	0.36	0.33	0.26	0.22	0.20	0.14	0.11	0.10
$[1,5)$	P_∞	0.43	0.38	0.35	0.33	0.28	0.26	0.18	0.20	0.18
	P_1	0.40	0.34	0.29	0.31	0.25	0.22	0.16	0.18	0.15
	P_5	0.42	0.37	0.34	0.33	0.28	0.25	0.18	0.19	0.17
	P_{10}	0.43	0.38	0.35	0.33	0.28	0.26	0.18	0.20	0.18
$[5,10)$	P_∞	0.53	0.39	0.31	0.38	0.24	0.23	0.20	0.14	0.15
	P_1	0.40	0.31	0.23	0.32	0.20	0.18	0.17	0.11	0.12
	P_5	0.45	0.36	0.27	0.36	0.24	0.20	0.19	0.14	0.14
	P_{10}	0.48	0.38	0.29	0.36	0.24	0.21	0.19	0.14	0.14

Table 13: Absolute Improvements when $P_x \neq P_0$, *CD*

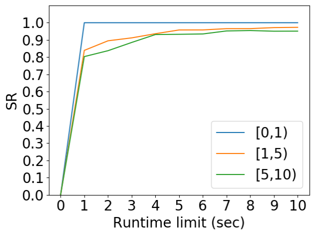
		25%			50 %			75 %		
		k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
$[0,1)$	P_∞	0.47	0.44	0.41	0.34	0.36	0.32	0.18	0.21	0.19
$[1,5)$	P_∞	0.22	0.21	0.21	0.22	0.20	0.20	0.14	0.14	0.13
	P_1	0.20	0.17	0.18	0.20	0.17	0.17	0.13	0.12	0.12
	P_5	0.22	0.19	0.19	0.21	0.18	0.18	0.13	0.13	0.12
	P_{10}	0.22	0.21	0.20	0.21	0.19	0.19	0.13	0.13	0.13
$[5,10)$	P_∞	0.34	0.31	0.30	0.30	0.27	0.25	0.18	0.17	0.15
	P_1	0.28	0.26	0.25	0.24	0.24	0.22	0.15	0.14	0.14
	P_5	0.30	0.26	0.26	0.26	0.24	0.23	0.16	0.15	0.15
	P_{10}	0.30	0.28	0.27	0.27	0.25	0.23	0.16	0.15	0.15

bound on the arrival probability. Then, SR represents the ratio of how much P_x can improve wrt. the largest possible improvement.

We construct an SR performance profile for Algorithm 4 on *DK* to examine how the probability of arrival within budget t improves as runtime limit x increases. Figures 8(a–b) show that as x increases, the improvement ratio increases. The sharpest improvement is from 0 to 1 second; and after 1 second, the improvement is slow. This is consistent with the observations when using path similarity (cf. Figure 7). This suggests that the problem studied is well suited for an anytime solution.



(a) Time budget 25%



(b) Time budget 75%

Figure 8: SR performance profile, *DK*, $k = 2$.

Next, we examine the absolute arrival probability improvements on anytime routing results—the difference in probability of arrival within t between P_x and P_0: $Prob(P_x, t) - Prob(P_0, t)$.

Tables 12 and 13 show that the absolute differences are large, suggesting that there is a strong cost dependence, and paths P_x exist that are significantly better than P_0. When changing the budget from 25% to 75%, the average differences decrease for both data sets. This is due to the probability of P_0 being at least as large as the budget percentage, and thus offering less space to improve the probability with a large budget percentage. We also observe that increasing the value of k generally leads to a slightly lower probability improvement as queries become slower and fewer finish by x. However, the differences are small, suggesting low sensitivity to k when using the hybrid cost estimation.

Tables 7 and 12 show that we sometimes find a path P_x that is different from both P_0 and P_∞. For example, for *DK*, in the $[1, 5)$ category with a 25% budget and $k = 2$, P_5 has a lower average probability difference than P_∞ in Table 12; yet, according to Table 7, P_5 and P_∞ have the same number of paths that are different from P_0 with this configuration. This suggests that cases exist where the anytime algorithm finds a path P_x that is better than P_0, but is still not the best.

Examining P_x vs. P_∞ for both data sets, we find that 5 seconds is an acceptable limit for queries in distance category $[1, 5)$ while queries in category $[5, 10)$ need at least 10 seconds. However, with a large budget, it is also more acceptable to decrease x. For example, for 75% in $[5, 10)$, we obtain the same result with P_5 as with P_{10}, no matter the data set, suggesting that time limits should be stated relative to budget sizes.

We conclude that the use of hybrid routing is attractive in road networks with cost dependence. Further, it is acceptable to limit the worst case execution time by using an anytime variant of Algorithm 4.

5.2.3 *Time Dependence*

We proceed to explore how to support time dependence in the Hybrid Model. We split a day into two time periods, peak periods including 7:00–9:00 and 15:00–17:00 and off-peak periods including all remaining time. We separate the trajectories according to the peak and off-peak periods, and we further split each trajectory set into a training set (80%) and a test set (20%). Next, we build two sets of regression models and binary classifiers using the peak training set and off-peak training set, respectively. Table 14 reports the KL-divergence when using the two models to estimate path cost distributions given the peak and off-peak test sets.

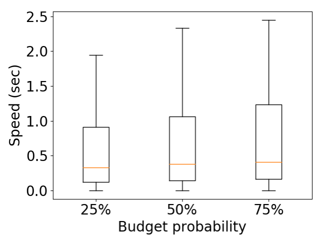
Table 14: KL-divergence of Time Dependent Models

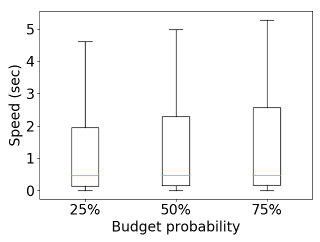
	Peak testing data	Off-peak testing data
Peak model	0.16	0.27
Off-peak model	0.21	0.14

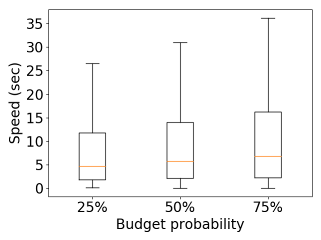
The results show that the time-specific models built on data from different periods can offer increased accuracy. However, two challenges make the integration of time dependence non-trivial. First, when building time dependent models, we may not have sufficient data to train the models, e.g., when training a model for every 15 minutes. Second, peak periods may be different across edges. Using global peak and off-peak periods may yield inaccurate peak and off-peak models. For example, we may include training data from edges with no peaks when building the peak model. Thus, it is non-trivial to support time dependence when using regression-based estimation models. This requires sophisticated designs, e.g., use of transfer learning or multi-task learning [13, 14]. We leave this as future work.

5.2.4 *Efficiency without Anytime Routing*

In the final experiments, we study the efficiency when not using anytime routing. We study the runtime of computing P_∞, the time needed to obtain the optimal path. Although the worst-case running time can be very long, it does not limit the practicality of the proposed hybrid routing since we can terminate the algorithms at any time and get good enough results as shown in Section 5.2.2.


(a) $[1, 5)$, $k = 2$


(b) $[1, 5)$, $k = 4$


(c) $[5, 10)$, $k = 2$

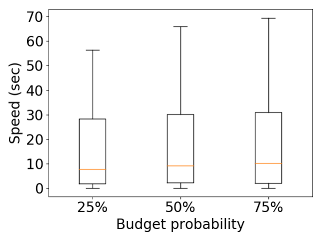

(d) $[5, 10)$, $k = 4$

Figure 9: Efficiency without anytime routing, *CD*.

Figures 9(a-d) show boxplots of the running times of computing P_∞ with Algorithm 4 when using different query distance categories, values of k, and time budgets. The whiskers represent 5th and 95th percentiles. We omit figures for distance category $[0, 1)$ as all short queries terminate in less than a second, no matter the configuration. We also omit figures for $k = 3$ as they offer similar results.

Examining the results for longer distance queries reveals that larger budgets lead to an increased variance in execution time. Given the same budget, varying k has a significant effect on the run time. Larger k offer less pruning potential and thus requires longer run time.

For all configurations, the mean running time exceeds the median substantially due to a few slow queries. There can be several reasons for this. First, query pairs are categorized w.r.t. their Euclidean distances, while shortest path distances always exceed the Euclidean distances. This may affect the search space, yielding a much larger set of potential paths than what is typical for the query category. For example, near a fjord or a river, we may need to take a detour to cross a bridge. This is particularly the case for the long category queries, where P_∞ sometimes has a length exceeding $15\ km$.

Second, varying numbers of measurements on edges may yield loose cost bounds. This happens in cases with many traversals on edges. Here, outlier measurements of traversals at very low speeds give some edges a very small probability of having a very high traversal cost. This yields a larger search space for Algorithm 4. We have applied simple heuristics to remove obvious outliers, but some outliers may still exist, which calls for the use of advanced outlier detection methods [15, 16].

Third, the use of NN models to estimate cost distributions is expensive. The impact of this can be seen in Figures 9 (c) and (d), when k is increased from 2 to 4, it greatly increases the frequency of regression-based estimation, and almost doubles the worst case time it takes to obtain P_∞.

6. RELATED WORK

Travel Cost Modelling. Extensive research has been conducted on modeling travel costs such as travel time, greenhouse gas emission [17], and fuel consumption [18], and some methods employ machine learning [19–21]. However, most of these studies consider only *deterministic costs*, e.g., average travel times. Instead, we focus on stochastic travel costs. It is non-trivial to extend existing studies to support stochastic travel costs. In addition, existing methods consider a setting where a path is given and thus do not satisfy the incremental property. This means that existing path cost estimation methods do not fit the setting of a routing algorithm. A study [22] considers stochastic costs, but only works for individual edges not for paths.

Stochastic travel costs have also been studied, but often assuming cost independence [23–25]. Some studies consider traversal costs as a function of time but assume no cost dependence between neighboring edges if a departure time is given [6, 26, 27]. This paper's cost model considers spatial dependence, i.e., adjacent edges may be cost dependent. One study integrates spatial dependence into the cost model by examining historical trajectories to reuse path costs [8], but it is only able to model cost dependence if trajectories exist that cover two or more consecutive edges in the path. Our approach also relies on trajectories to model spatial dependence, but it does not need trajectories that follow the path for which a cost is computed. Another study also models spatial dependence between edges [28]. This approach relies on assumptions such as turn speed bounds, and it neither utilizes real-world costs nor considers stochastic costs. In contrast, we make no assumptions about which elements affect spatial dependence. Finally, a study models spatial dependence between adjacent edges [29], but assumes that all pairs of adjacent edges have a known joint distribution. To the best of our knowledge, we are the first to propose a cost model that combines convolution and machine learning to approximate path cost distributions more accurately.

Stochastic Routing. Existing stochastic routing algorithms often assume that edge cost distributions are independent and perform pruning based on stochastic dominance [6, 23, 27]. A few studies consider cost dependence [30–32], but they only use stochastic dominance-based pruning if two edges are independent and thus inefficient. We utilize an anytime algorithm design that provides good results within a runtime limit. In addition, to achieve efficiency, we propose two additional pruning techniques using pivot paths and A* like optimistic costs. The present paper offers a substantially extended coverage over an earlier four-page report [7].

7. CONCLUSION AND FUTURE WORK

We propose means of stochastic routing together with a hybrid model for path cost computation. We first show that it is beneficial to use machine learning for path cost computation because this enables the capture of cost dependencies among the edges in paths. Next, we propose a hybrid model that computes costs, and then we integrate this model into an anytime routing algorithm. We conduct extensive experiments that offer insight into the efficiency and result quality achieved by the algorithm.

In future work, it is of interest to attempt to consolidate different NNs into a single model using to avoid the process of model selection and to employ advanced learning models that better capture road network topology, e.g., graph convolution networks [33, 34]. In addition, it is of interest to consider context aware routing [35], such as time-dependent routing and personalized routing.

Acknowledgments

This research was supported in part by Independent Research Fund Denmark under agreements 8022-00246B and 8048-00038B, and by the DiCyPS center, funded by Innovation Fund Denmark.

8. REFERENCES

[1] Jilin Hu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. Risk-aware path selection with time-varying, uncertain travel costs: a time series approach. *VLDB J.*, 27(2):179–200, 2018.

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

[3] Chenjuan Guo, Bin Yang, Jilin Hu, and Christian S. Jensen. Learning to route with sparse trajectory sets. In *ICDE*, pages 1073–1084, 2018.

[4] Chenjuan Guo, Christian S. Jensen, and Bin Yang. Towards total traffic awareness. *SIGMOD Record*, 43(3):18–23, 2014.

[5] Jilin Hu, Bin Yang, Christian S. Jensen, and Yu Ma. Enabling time-dependent uncertain eco-weights for road networks. *GeoInformatica*, 21(1):57–88, 2017.

[6] Bin Yang, Chenjuan Guo, Christian S. Jensen, Manohar Kaul, and Shuo Shang. Stochastic skyline route planning under time-varying uncertainty. In *ICDE*, pages 136–147, 2014.

[7] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. A hybrid learning approach to stochastic routing. In *ICDE*, pages 2010–2013, 2020.

[8] Jian Dai, Bin Yang, Chenjuan Guo, Christian S. Jensen, and Jilin Hu. Path cost distribution estimation using trajectory data. *In PVLDB*, 10(3):85–96, 2016.

[9] C. C. Heyde. *Central Limit Theorem*. Wiley StatsRef: Statistics Reference Online, 2014.

[10] A. Arun Prakash. Pruning algorithm for the least expected travel time path on stochastic and time-dependent networks. *Transportation Research Part B: Methodological*, 108:127 – 147, 2018.

[11] Simon Aagaard Pedersen, Bin Yang, and Christian S Jensen. Fast stochastic routing under time-varying uncertainty. *VLDBJ*, online first, 2019.

[12] Huiping Liu, Cheqing Jin, Bin Yang, and Aoying Zhou. Finding top-k shortest paths with diversity. *In IEEE TKDE*, 30(3):488–502, 2018.

[13] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. Distinguishing trajectories from different drivers using incompletely labeled trajectories. In *CIKM*, pages 863–872, 2018.

[14] Razvan-Gabriel Cirstea, Darius-Valer Micu, Gabriel-Marcel Muresan, Chenjuan Guo, and Bin Yang. Correlated time series forecasting using multi-task deep neural networks. In *CIKM*, pages 1527–1530, 2018.

[15] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. Outlier detection for time series with recurrent autoencoder ensembles. In *IJCAI*, pages 2725–2732, 2019.

[16] Tung Kieu, Bin Yang, and Christian S. Jensen. Outlier detection for multidimensional time series using deep neural networks. In *MDM*, pages 125–134, 2018.

[17] Chenjuan Guo, Bin Yang, Ove Andersen, Christian S. Jensen, and Kristian Torp. Ecomark 2.0: empowering eco-routing with vehicular environmental models and actual vehicle fuel consumption data. *GeoInformatica*, 19(3):567–599, 2015.

[18] Chenjuan Guo, Bin Yang, Ove Andersen, Christian S. Jensen, and Kristian Torp. Ecosky: Reducing vehicular environmental impact through eco-routing. In *ICDE*, pages 1412–1415, 2015.

[19] Dong Wang, Junbo Zhang, Wei Cao, Jian Li, and Yu Zheng. When will you arrive? estimating travel time based on deep neural networks. In *AAAI*, pages 2500–2507, 2018.

[20] Zheng Wang, Kun Fu, and Jieping Ye. Learning to estimate the travel time. In *SIGKDD*, pages 858–866, 2018.

[21] Sean Bin Yang and Bin Yang. Learning to rank paths in spatial networks. In *ICDE*, pages 2006–2009, 2020.

[22] Bin Yang, Chenjuan Guo, and Christian S. Jensen. Travel cost inference from sparse, spatio-temporally correlated time series using markov models. *In PVLDB*, 6(9):769–780, 2013.

[23] Evdokia Velinova Nikolova, Matthew Brand, and David R Karger. Optimal Route Planning under Uncertainty. *ICAPS*, pages 131–141, 2006.

[24] Anthony Chen and Zhaowang Ji. Path finding under uncertainty. *Journal of Advanced Transportation*, 39(1):19–37, 2005.

[25] Ajith B. Wijeratne, Mark A. Turnquist, and Pitu B. Mirchandani. Multiobjective routing of hazardous materials in stochastic networks. *European Journal of Operational Research*, 65(1):33–43, 1993.

[26] Mohammad Asghari, Tobias Emrich, Ugur Demiryurek, and Cyrus Shahabi. Probabilistic estimation of link travel times in dynamic road networks. In *SIGSPATIAL*, pages 47:1–47:10, 2015.

[27] Michael P. Wellman, Matthew Ford, and Kenneth Larson. Path planning under time-dependent uncertainty. In *UAI*, pages 532–539, 1995.

[28] Robert Geisberger and Christian Vetter. Efficient routing in road networks with turn costs. In *International Symposium on Experimental Algorithms*, pages 100–111. Springer, 2011.

[29] Ming Hua and Jian Pei. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *EDBT*, pages 347–358, 2010.

[30] Bin Yang, Jian Dai, Chenjuan Guo, Christian S. Jensen, and Jilin Hu. PACE: a path-centric paradigm for stochastic path finding. *VLDBJ*, 27(2):153–178, 2018.

[31] Georgi Andonov and Bin Yang. Stochastic shortest path finding in path-centric uncertain road networks. In *MDM*, pages 40–45, 2018.

[32] Saad Aljubayrin, Bin Yang, Christian S. Jensen, and Rui Zhang. Finding non-dominated paths in uncertain road networks. In *ACM SIGSPATIAL*, pages 15:1–15:10, 2016.

[33] Jilin Hu, Chenjuan Guo, Bin Yang, and Christian S. Jensen. Stochastic weight completion for road networks using graph convolutional networks. In *ICDE*, pages 1274–1285, 2019.

[34] Jilin Hu, Chenjuan Guo, Bin Yang, Christian S. Jensen, and Hui Xiong. Stochastic origin-destination matrix forecasting using dual-stage graph convolutional, recurrent neural networks. In *ICDE*, pages 1417–1428, 2020.

[35] Chenjuan Guo, Bin Yang, Jilin Hu, Christian S. Jensen, and Lu Chen. Context-aware, preference-based vehicle routing. *VLDBJ*, online first, 2020.