A Hybrid Learning Approach to Stochastic Routing

Master's thesis (4+4)
Simon Aagaard Pedersen
Software Engineering

AALBORG UNIVERSITY
STUDENT REPORT

Computer Science
Aalborg University
http://www.aau.dk

Title:
A Hybrid Learning Approach to Stochastic Routing

Theme:
Scientific Theme

Project Period:
Spring Semester 2019

Participant(s):
Simon Aagaard Pedersen

Supervisor(s):
Christian S. Jensen
Bin Yang

Copies: 1

Page Numbers: 16

Date of Completion:
16th August 2019

Project Summary

Routing is a central component of many GPS-based applications, and an increasingly large amount of historical routing data is becoming available. This data can be applied to modern routing models and algorithms to enable a better approximation of real-life road networks. This paper investigates how historical data in combination with machine learning can be applied to accurately estimate traversal costs and integrates the resulting estimation technique into a stochastic graph network model called the Hybrid Model to more accurately capture the mobility of real-life road networks. The model encodes road intersections as vertices and road segments as edges, and each edge is associated with a distribution that represents a traversal cost of the edge. Following the construction of the model, we examine why traditional routing algorithms such as Dijkstra's algorithm cannot be applied in the model, and we propose a routing algorithm, called Hybrid Search, that enables stochastic routing.

The Hybrid Model utilizes machine learning to estimate the cost of traversing a path consisting of several edges. In stochastic modeling, a traditional approach to calculating the cost of traversing a path consisting of multiple edges is to use convolution. However, convolving two distributions is done under the assumption that they are independent, i.e., observing a cost in one distribution does not change the probability of any cost in the second distribution. This assumption is a simplification, and it does not hold in real-life road networks. For instance, the traversal cost of a road segment immediately after an intersection may depend on the traversal cost observed on the road segment leading up to the intersection: stopping at the intersection implies deceleration before the intersection and acceleration after the intersection, while not stopping yields lower travel total time.

In order to capture traversal cost dependencies, we employ a Neural Network (NN) to estimate the sum of two distributions. We do this by utilizing a large amount of historical traversal records as well as extracting a number of auxiliary road network properties that intuitively may impact the degree of cost dependence between the traversals of two adjacent road segments. As such, we build a machine learning model that accepts as input the stochastic traversal costs for two road segments as well as secondary properties such as the existence of traffic signals between edges and the degree that a vehicle has to turn to move from one road segment to the next. We find that the NN in many cases estimates a cost that is more similar to the ground truth than does traditional convolution, but we also note that this is not always the case because the dependence may be weak. Consequently, we construct a classifier that is used to determine whether we should utilize convolution or NN to compute a cost. The Hybrid Model thus follows two simple steps to compute the summed cost of two distributions: First, it uses the classifier to determine whether convolution or NN is expected to produce the most accurate result. Next, it applies the best approach to compute a result. We offer empirical evidence that the Hybrid Model produces better results than always using either NN or convolution.

While we wish to compute costs for paths of arbitrary length during routing, the Hybrid Model takes its outset in the computation of the cost of traversing two adjacent edges. We provide empirical evidence that suggests that NN-based estimation should be repeated at most four times, i.e., for paths consisting of four vertices, as NN-based estimation becomes less accurate than convolution for longer paths. We proceed to extend the Hybrid Model with two path-cost building algorithms: One that accepts an arbitrary number of edges and constructs a distribution representing the cost of traversing the path that consists of all edges, and one that accepts a path and a single edge and computes the cost of traversing the edge after traversing the path. These algorithms are essential components for the Hybrid Model to enable stochastic routing in a graph network.

The paper proceeds to solve the problem of finding the best path in a stochastic graph network given a time budget. Stochastic routing is traditionally performed under the assumption that the employed graph network adheres to a property called subpath optimality. Given the shortest path between a source and destination, this property states that all subpaths that can be constructed using this shortest path are also shortest paths. As a consequence, the property enables heavy pruning of the search space, which can yield much reduced execution times. However, this property does not hold when performing routing with the Hybrid Model. We propose a stochastic routing algorithm, called Hybrid Search, that uses several pruning techniques to reduce the search space. First, we employ an A*-like cost reach to reject potential paths given a time budget. Second, we maintain a pivot variable that always represents the best path found. This variable enables evaluation of all paths based on their cost and the A* cost estimate for reaching the destination vertex. Third, we determine that we can prune paths using stochastic dominance whenever they reach a node that is fully cost independent, i.e., no edge entering the node is cost dependent on any edge exiting the node.

Finally, we present an anytime extension of the Hybrid Search that, given an execution time limit, will return the best possible path that can be found within this time limit. We show through empirical studies that reasonable execution time limits produce results that, on average, are very similar to those produced by the algorithm without a time restriction. This suggests that Hybrid Search quickly finds a good path, but may spend substantial additional time in order to find the best path, which may be only marginally better. We also show that utilizing the Hybrid Model produces paths that often are different from the path found using Dijkstra's algorithm, suggesting that intra-city routing may benefit from considering cost dependence.

CONTENTS

A Hybrid Learning Approach to Stochastic Routing

Simon Aagaard Pedersen
Department of Computer Science
Aalborg University
Aalborg, Denmark
sape@cs.aau.dk

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

Christian S. Jensen
Department of Computer Science
Aalborg University
Aalborg, Denmark
csj@cs.aau.dk

***Abstract*—Increasingly available trajectory data enables detailed capture of traffic conditions. We consider an uncertain road network graph, where each graph edge is associated with a travel time distribution, and we study probabilistic budget routing that aims to find the path with the highest probability of arriving within a given time budget. In this setting, a fundamental operation is to compute the travel cost distribution of a path from the cost distributions of the edges in the path. Solutions that rely on convolution generally assume independence among the edges' distributions, which often does not hold and thus incurs poor accuracy. We propose a hybrid approach that combines convolution and machine learning based estimation to take into account dependencies among distributions in order to improve accuracy. Next, we propose an efficient routing algorithm that is able to utilize the hybrid approach and that features effective pruning techniques to enable efficient routing. Empirical studies on a substantial real world trajectory set offer insight into the properties of the proposed solution, indicating that it is practical.**

I. Introduction

Emerging disruptive innovations in transportation, e.g., autonomous vehicles and transportation-as-a-service, call for high-resolution routing where traffic uncertainty is captured accurately. For example, when an autonomous taxi needs to arrive at an airport within a deadline, having accurate travel time distributions of candidate paths enables the taxi to choose the most reliable path. In the example in Table I, if the deadline is within 60 minutes, path P_1 is more reliable than path P_2, since P_1 gives a 0.9 probability of arriving within 60 minutes, which exceeds P_2's probability of 0.8. When using average travel times, the taxi will choose P_2 because P_2 has average travel time 51, while P_1's average travel time is 53. Thus, the taxi has a higher risk of being late.

Table I: Travel Time Distributions of Two Paths to the Airport

Travel time (mins)	[40, 50)	[50, 60)	[60, 70)
P_1	0.3	0.6	0.1
P_2	0.6	0.2	0.2

Next, more and more trajectories are becoming available that capture the movements of vehicles and traffic conditions. This provides a solid data foundation for high-resolution traffic uncertainty modeling. We often 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 the edges' corresponding trajectory pieces. The edge weights, i.e., their travel time distributions, are often assumed to be independent to each other. Then the travel time distribution of a path is computed based on the convolution of the travel time distributions of the edges in the path.

However, the travel times on edges are often dependent due to a number of factors, e.g., traffic lights and turns. As a result, the independence assumption often leads to inaccurate results. For example, consider a path that consists of only two edges, e_1 and e_2. Assume that 100 trajectories cover the path. Of these, 50 traverse e_1 in 10 seconds and e_2 in 20 seconds, yielding a total traversal time of 30 seconds. The remaining 50 trajectories traverse e_1 in 15 seconds and e_2 in 25 seconds, yielding a total traversal time of 40 seconds. A driver then either traverses both edges quickly or slowly, and it is very unlikely that a driver traverses one edge fast and the other slow, or vice versa.

Next, we split the trajectories to fit edges e_1 and e_2 such that we obtain cost distributions H_1 and H_2 of the two edges, as shown in Table II. The convolution of H_1 and H_2, shown in Table III, does not reflect the same reality as do the trajectories. Rather, we now have a large probability of traversing the two edges in 35 seconds, which no trajectories support.

Table II: Distributions for e_1 and e_2

H_1		H_2	
Travel Time	Probability	Travel Time	Probability
10	0.5	20	0.5
15	0.5	25	0.5

To better capture travel time dependency, we propose a hybrid learning approach. First, we train a regression model, specifically a neural network, that is able to take into account

Table III: e_1 and e_2 convolved

Travel Time	Probability
30	0.25
35	0.50
40	0.25

Table IV: Ground truth for $P = \langle e_1, e_2 \rangle$

Travel Time	Probability
30	0.50
40	0.50

two edges' distributions and a number of road condition features that describe the relationships between the two edges, e.g., whether a traffic light is in-between the two edges or the angle between the two edges, while estimating the travel time distribution of the path that consists of the two edges. In our example, when training the regression model, the input is H_1, H_2, and road condition features. The ground truth for the input is distribution H_P, shown in Table IV.

Second, we observe that convolution is not always a bad choice, especially when the dependency between two edges is weak. Thus, we train a binary classifier to judge whether we should use convolution or the trained regression model. The resulting hybrid learning approach yields better accuracy compared to using only convolution or only regression.

We proceed to investigate the integration of the proposed hybrid learning approach into a routing algorithm to support stochastic routing, specifically probabilistic budget routing. In routing algorithms, we often need to compare two paths P_1 and P_2 that connect the same pair of vertices. In a deterministic setting, if the cost of P_1 is smaller than that of P_2, we can disregard P_2 since any path P' that is extended from P_2 has a higher cost than the path where P_1 is used instead of P_2.

In a stochastic setting, paths have travel time distributions, not deterministic values. Let P_1 and P_2 have distributions D_1 and D_2, respectively. We often use stochastic dominance to compare two such distributions. In short, if distribution D_1 stochastically dominates 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 $H_1 \oplus H_e$ of path $P_1' = \langle P_1, e \rangle$ dominates the distribution $H_2 \oplus H_e$ of path $P_2' = \langle P_2, e \rangle$, where $\oplus$ denotes convolution. However, when using the regression model $RM(\cdot, \cdot)$, we cannot guarantee that $RM(H_1, H_e)$ dominates $RM(H_2, H_e)$ when H_1 dominates H_2. This resulting inability to prune based on stochastic dominance makes routing more challenging.

We propose a routing algorithm that employs the hybrid learning approach to estimate path travel time distributions along with additional speed-up techniques to ensure efficiency. The proposal includes a method to estimate best possible distributions for paths, enabling an A*-like heuristic, and using so-called pivot paths to enable additional pruning. Finally, we provide an anytime extension of the algorithm that can deliver a solution at any time; and the longer the algorithm runs, the more accurate the solution becomes. This provides flexibility on how long a user may want to wait.

To the best of our knowledge, this is the first study that integrates machine learning based cost estimation with a routing algorithm. We make four specific contributions. We propose (i) a hybrid learning approach to accurately estimate the travel time distributions of paths, (ii) a routing algorithm that employs the hybrid learning approach to support probabilistic budget routing, and (iii) an anytime query processing extension to the routing algorithm, and (iv) we conduct a comprehensive empirical study to justify our design choices and to offer insight into the proposed methods.

The remainder of the paper is organized as follows: Section II presents preliminaries and the problem. Section III presents a hybrid approach for estimating travel cost distribution of a path. Section IV presents a routing algorithm that utilizes the proposed hybrid approach. Section V presents the empirical study. Finally, Section VI covers related work, and Section VII concludes and suggests future work.

II. Preliminaries

A. Basic Concepts

1) Road Networks: A road network is represented as a graph $G = (V, E, C)$, where V is a set of vertices, E is a set of edges of the form $e_i = (v_j, v_k)$, and C is a cost function $C : E \rightarrow D$ that maps edges to their corresponding traversal cost distributions.

A path $P = \langle e_1, \ldots, e_n \rangle$ is a sequence of edges such that $\forall_{e \in P}\ (e \in E)$. We only consider simple paths, meaning all vertices covered by the path are distinct. A subpath P_S of a path P is a path consisting of a contiguous subsequence of the edges in P. A pre-path P^{-n} of a path P is P with the last n edges removed. Consequently, $|P^{-n}| = |P| - n$. Path expansion occurs when a path P is expanded by an edge e, denoted $P' = \langle P, e \rangle$.

2) Uncertain road networks: We use equi-width histograms to represent distributions. A histogram is a collection of $(bucket, probability)$ pairs with each $bucket$ representing a cost range. The sum of all probabilities in a histogram sum to 1.0, and the probability for all values within each bucket is uniform. We use histograms because they can represent arbitrary distributions and are more compact than Gaussian mixture models [18].

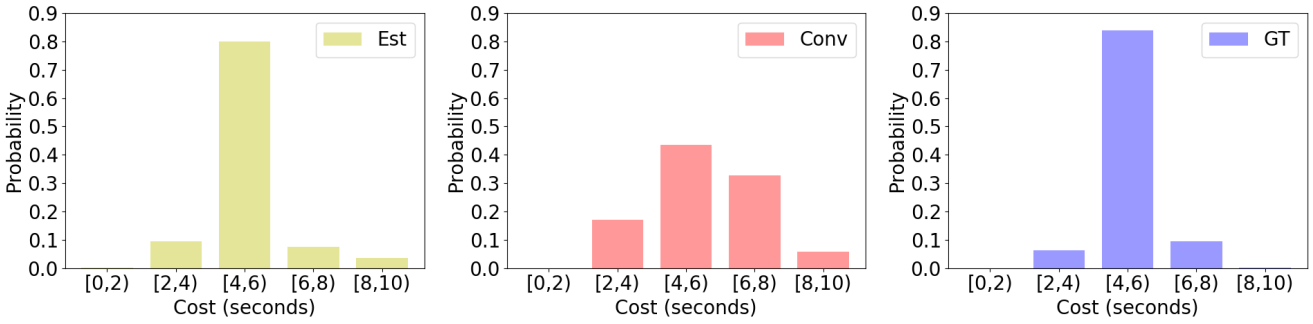
We consider traversal time with the lowest unit of measure being a second. To obtain a histogram for an edge, we build a probability distribution D based on the traversal time values. Next, a bucket width w, a lower cost bound lb, and an upper cost bound ub are chosen, yielding a histogram with $\lceil (ub - lb)/w \rceil$ $(bucket, probability)$ pairs. Then, we place all probabilities in D in their appropriate bucket such that the probability of each pair represent the sum of all probabilities of the bucket's costs in D.

B. Path costs

The cost of a path can be represented by the sum of the edge costs. When costs are represented by discrete distributions, we can use circular convolution to sum the costs with the underlying assumption that all distributions are independent [16], [19]. The convolution of two independent discrete distributions X and Y, denoted $Z = X \oplus Y$, can be expressed as $Z(z) = \sum_{x \in X} (f_X(x) \cdot f_Y(z - x))$.

C. Problem Definition

Probabilistic Budget routing: Given a source s, a destination d, and a time budget t, we aim to select 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 $PBR(s, d, t) = \text{argmax}_{P \in \mathbb{P}}\ Prob(cost(P) \leq t)$.



(a) Estimation (b) Convolution (c) Ground Truth

Figure 1: Example based on real world data

III. Hybrid Learning Model Building

A. Limitations of Convolution

The result of applying convolution to two distributions is only accurate if the distributions are independent. However, in road networks this is often not the case [6]. Rather, cost dependence is a common phenomenon. For instance, a traffic light has the potential to create a dependence: cars either decelerate, stop, and then accelerate in case of a red light, or cars pass through at unchanged speed in case of a green light—the other two combinations of behaviors before and after the traffic light are not likely.

Next, due to the central limit theorem [10], repeatedly convolving independent distributions eventually results in distributions akin to Gaussian distributions, which often reflects reality poorly [6], [19].

The variance of the sum of two normally distributed random independent variables can be derived as the sum of the variance of the individual random variables. With repeated convolution, we therefore obtain distributions with increasingly large variance, reducing the possibility of any spikes in traversal costs, and instead moving towards a distribution that is flatter and more uniform between the lower and upper bounds.

Next, convolution using discrete histograms with bucket width $w > 1$ is not good at capturing spikes. This is due to the lost granularity when discretizing values into equi-width buckets. As an example, Figures 1(a–b) shows a real-world example of the difference between convolution and the ground truth. The convolution result is dissimilar to the ground truth because it cannot obtain the same spike; even if we were to convolve two histograms each consisting of the same cost pair, e.g., $([2, 4), 1.0)$, we would obtain the result $\{([4, 6), 0.5), ([6, 8), 0.5)\}$. In general, for a discrete convolution with uniformly distributed equi-width buckets of width $w > 1$, no bucket in the output can have a probability of 1.0, and probabilities above 0.5 are rare.

B. Learning Path Cost Distributions

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

1) Short paths: We first consider *short paths*, i.e., paths with two edges. We treat path cost estimation as a regression problem: $\hat{H}_P = F(H_1, H_2, C)$, where H_1 and H_2 are cost 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. Regression function F estimates the cost distribution $\hat{H}_P$ of path $P = \langle e_1, e_2 \rangle$. We proceed to elaborate on how to prepare training data and on the regression function F.

We employ GPS trajectories for training. In particular, we identify short paths that are traversed frequently by trajectories. Specifically, we use the 5000 most traversed edge pairs in our road network, which each has from 5603 to 295 unique trajectory samples that represent a ground truth cost distribution for their full traversal.

We then split these 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 $P = \langle e_1, e_2 \rangle$. This method applies to adjacent edges with no common trajectories, as long as we can derive a distribution H for both edges.

Next, we identify features C that characterize the two edges. These include the lengths and speed limits; the angle between the edges; whether there is a traffic light between the edges; the minimum, expected, and maximum traversal times for the edges; and the road types (e.g., highway) of the edges. The intuition is that the degree of dependence between two edges may be affected by these factors that thus should be considered by the regression model. Some features are discretized and one-hot encoded, whereas others are floating point values.

We choose to use a classic multilayer perceptron neural network (NN) as the regression model since its ability to capture non-linear relationships among inputs is essential for capturing distribution correlations. We require that H_1, H_2, and H_P are homogeneous histograms, each having n $(bucket, probability)$ pairs, and C has m features. Thus, the input layer has $2 \cdot n + m$ neurons. The first $2 \cdot n$ neurons correspond to the probabilities of the n cost pairs in H_1 and

H_2, and the last m neurons correspond to the features in C. We input a vector of probabilities without buckets, i.e., cost ranges. The output layer has n neurons that correspond to the estimated probabilities of the n buckets, i.e., $\hat{H}_P$. 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 for edges e'_1 and e'_2 to the NN, which estimates a 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–c) show that using estimation instead of convolution can yield distributions that are more similar to the ground truth distribution. In this example, the KL-divergence between the ground truth and the estimation is 0.06, whereas the KL-divergence between the ground truth and the convolution is 0.23, i.e., estimation is best. This example demonstrates that machine learning techniques can compute more accurate representations of the sums of two dependent distributions.

Building multiple NNs: Since the input and the output histograms of the NN must be homogeneous, we train different NNs for different settings. We train different models to ensure the semantics of the input and output values stay the same, e.g., the n values in the output have the same meaning as each of the two n input values. We cannot create a single model that accepts arbitrarily large input distributions while maintaining a clear semantic meaning for each value. Instead, we build different models that each can process a fixed input space, but has a clear meaning. In particular, we consider four filtering parameters: 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, e.g., all training paths have a total distance within the distance bounds. This increases the likelihood that the selected training data is similar in structure, and we can use the distance bounds to identify the model to be used when given two edges as input. We also limit the lower and upper bounds on the costs. In combination with the distance bounds, this enforces the data to have reasonable driving speeds.

Another reason for utilizing cost bounds is that the NNs need homogeneous histograms, meaning that we have to determine the histogram structures of the input and output distributions beforehand. In practice this means that, given cost bounds and a bucket width, we can pre-calculate how many $(bucket, probability)$ pairs each histogram should be represented by. The output distribution is represented by the same bucket width and number of pairs. We choose to use a constant number of buckets for all histograms while varying the bucket width for each model. The reason is twofold: First, this yields the same number of buckets no matter how large an upper bound the input distributions have. We can accommodate large input distributions by using large buckets. Using instead a constant bucket width and varying number of buckets would have the effect that increasing the variance of the input distributions would require an increasingly large number of buckets, making the estimation inaccurate. Second, using a constant number of buckets eases model selection, as it is easy to select one of several models that only vary in the number of buckets and bucket width.

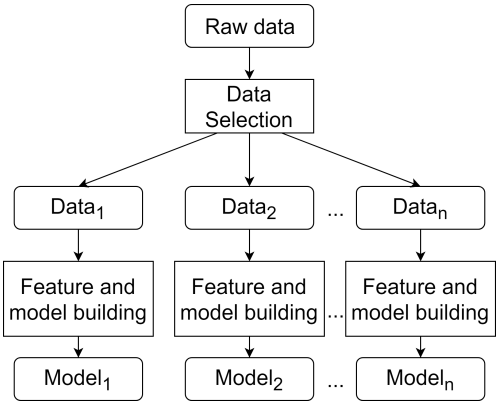


Figure 2: Building Cost Estimation Models for Short Paths

Model Selection Many different models can be built using varying filtering configurations, each with different strengths and weaknesses when compared to convolution. In general, we find that more specialized models produce better estimates, but specialized models also have limitations on their inputs, e.g., distance limitations cause samples outside the limits to be eliminated from the training set, thus reducing the size of the training set. Instead of creating a single model, we create a number of models with different configurations to ensure that any type of input can be processed by at least one model. Several models may be applicable for a given input, and we therefore have to provide means of selecting a good model.

Given input distributions and lengths, as well as several different estimation models with intersecting input spaces, we wish to select the model with the tightest fit to the input.

With this in mind, we select a model as follows: Given two input distributions D_1 and D_2 and two lengths L_1 and L_2, we first select the set of models, called S, trained on data with similar distance edges. Next, we select the model in S that minimizes the upper cost bound. When doing so, we ensure that the selected model has an upper bound that exceeds the sum of the largest measured costs of D_1 and D_2. This requirement makes it possible to obtain a distribution akin to the result of a convolution if the dependence is small.

Model performance It is not difficult to find cases where estimation outperforms convolution in terms of accuracy. We compare the two approaches when given identical input, e.g., if we estimate a distribution based on two input distributions given by $n = 5$ $(bucket, probability)$ pairs each, we perform convolution by first deriving the probabilities of each integer cost value covered by the buckets. We convert the convolution result back into a histogram with n equi-width buckets to enable comparison with the estimation result.

Figure 3 illustrates how different cost bounds with a constant bucket width affect the accuracy of estimation and convolution. It is clear that the accuracy of convolution deteriorates heavily with an increase in bucket width, whereas the accuracy of estimation improves.

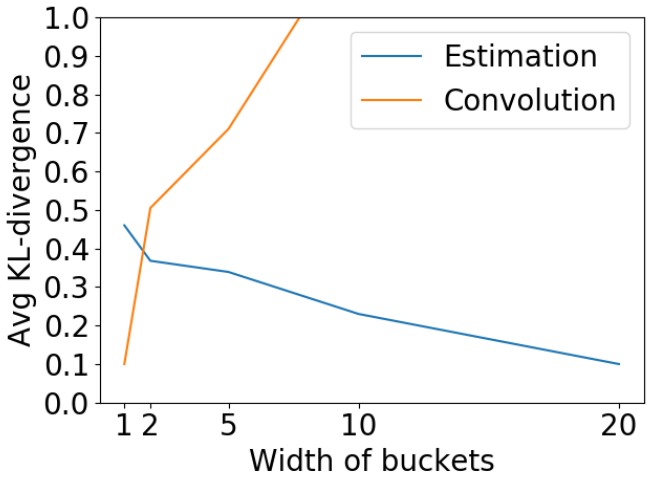


Figure 3: Result accuracy when varying bucket width with $n = 10$, and distance bounds $= (0, 100)$.

Figure 4 shows the probability of ML estimation vs. convolution being more similar to the ground truth for different bucket widths. Although the average KL-divergence grows as the bucket width increases for convolution, we still find that convolution outperforms estimation on a case-by-case basis 40% of the time for bucket width $b = 5$. The model used here is built using edge pairs no longer than 100 meters, which is a defining factor. If paths are exactly 100 meters and we have a bucket width of 10 seconds, the first bucket covers all trajectories with average speeds in the range $[36, \infty]$ km/h. As few roads have a speed limit below $36km/h$, we can expect the first bucket to have a near 1.0 probability. Recall that the model output distribution has the same number of buckets, bucket width, and cost bounds as the input. As mentioned in III-A, convolution cannot accurately capture such a discrete distribution; this causes convolution accuracy to deteriorate with increasing bucket size, while it becomes progressively easier for the estimation model to accurately estimate the ground truth because it tends towards the first bucket always having a probability of 1.0. However, even for bucket width b=2, we find that the estimation result has a higher probability of being more similar to the ground truth than the convolution result. This suggests that relying purely on either of the two will increase result inaccuracy. Furthermore, the average KL-divergence for convolution increases significantly with the bucket width, as seen in Figure 3, while the probability of being most similar decreases at a lower rate. This suggests that the convolution result accuracy has a very large variance and that the results tend to be either very good or very bad.

Figure 5 supports this hypothesis by visualizing the distribution of KL-divergence values for convolution and estimation

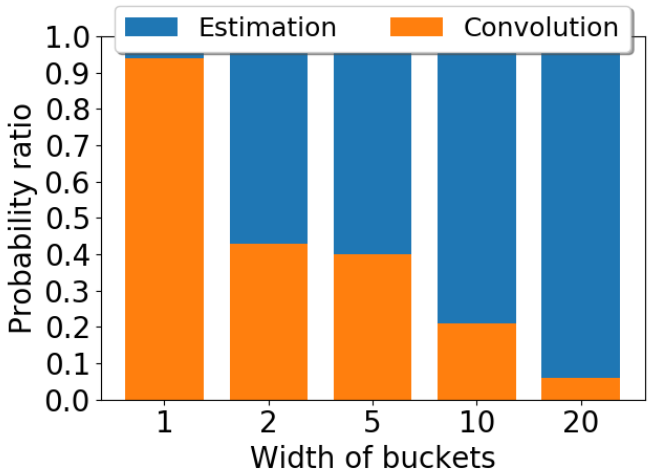


Figure 4: Likelihood of estimation and convolution being best when varying bucket width with $n = 10$ and distance bounds $= (0, 100)$.

across more than 500 edge pairs. Each edge pair has a significant number of trajectories traversing both edges sequentially, which we use to create ground truth distributions. Convolution yields larger accuracy spreads, and larger medians than does estimation.

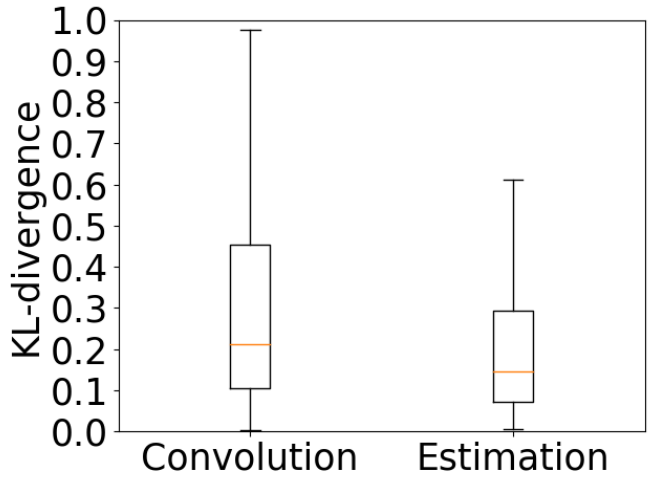


Figure 5: Boxplot of KL-divergence between results of convolution, estimation, and the ground truth. Based on more than 500 samples, with $n = 10$ and bucket width $b = 2$

2) Long Paths: When performing routing, we often need to compute cost distributions for paths that are longer than two edges. Path cost computation is an iterative process, as the cost of a path P can be computed by combining the cost of the pre-path P^{-1} up to the last edge with the cost of the last edge. We can use an NN model built for short paths to estimate the costs of longer paths by treating the pre-path as a "virtual" edge. However, with this approach, we can no longer expect a pair of edges to have common properties such as

similar expectation and variance. Instead, we need to consider the behavior of our NN model when the distributions of the two edges significantly differ.

Figure 6 shows the probability of repeated estimation vs. repeated convolution being superior for varying path lengths. It is clear that repeatedly applying an NN estimation model degrades the quality of the path cost and should not be done more than a few times before convolution is preferable.

The figure shows that for this particular model, estimation has a 59% probability of being best for paths consisting of four edges, whereas for paths of five edges, the probability is 22%. Thus, when computing the cost distribution of a long path, after applying the NN estimation 3 times, it is better to use convolution.

Other models display a similar behavior. We observe that for all estimation models, applying them repeatedly for cost computations increases the probability of obtaining results dissimilar to the ground truth than when using convolution.

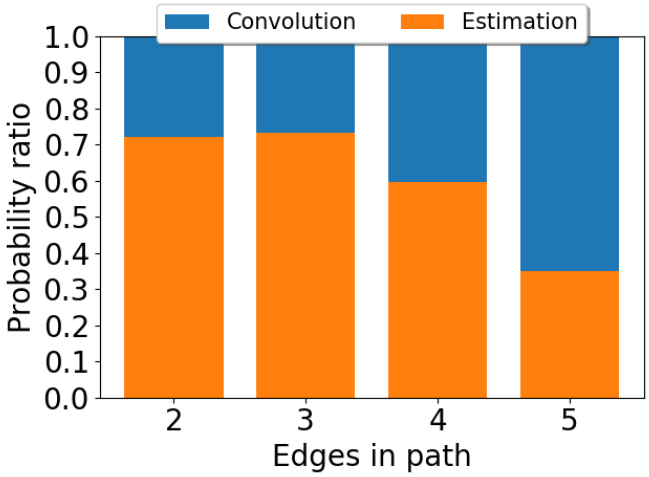


Figure 6: Probability ratio of repeated estimation or convolution being more similar to the ground truth for different path lengths

C. Hybrid Model

As discussed earlier, neither convolution nor estimation outperforms the other across all input configurations. Rather, both approaches have their advantages, depending of the cost dependencies and the granularity at which we wish to compute costs, i.e., bucket width, and path length.

To more accurately compute a path's cost distribution, we create a hybrid model by combining NN estimation and convolution. We introduce a Boolean classifier to determine which approach to use in a given context. During the model testing, where we determine how accurate a model is across a test data set, we cache a Boolean value encoding whether convolution or estimation is better for the given input. After terminating model fitting, we then build a binary classifier using logistic regression such that the likelihood scores of the output feature represent the estimated probabilities of whether convolution or estimation is most accurate.

The classifier functions in the same way as the estimation model, accepting a number of input features and outputting two scores that sum to 1 and represent the likelihoods of the two approaches being best. We use the same features for the classifier as we used for the estimation model. Figure 7 illustrates the process of using the classifier; it accepts two distributions D_1 and D_2, as well as a number road network properties encoded as features. The output scores returned by the classifier represent the likelihood of estimation or convolution being best. The closer the scores are to 0 and 1, the more confident we can be in the result. Figure 8 shows

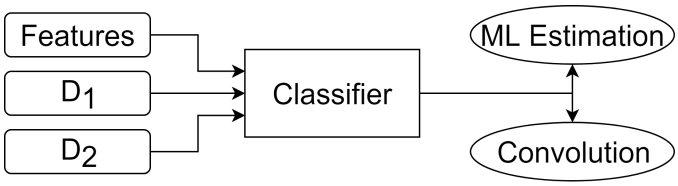


Figure 7: Using a classifier to choose between convolution and ML estimation

the accuracy of the hybrid approach vs. the accuracy of purely using estimation. 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 chooses wrong, the average difference in the likelihood scores is 0.12, and 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 obtain very similar result accuracy. Using the classifier, we label adjacent edge pairs as being dependent if estimation is to be used and independent if convolution is to be used.

The classifier only determines whether two adjacent edges are dependent or not. This is acceptable given the assumption that pair-wise 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 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.

Algorithm 1 builds the cost of a path of arbitrary length using the hybrid model. It first determines the set of independent edge pairs by traversing the path and using the classifier to determine whether convolution or estimation is better for each pair of adjacent edges. Then, it builds $i+1$ distributions, where i is derived from the number of independent edge pairs and the largest number of times we wish to use estimation. By using $len = k$ in line 7, we never repeat NN estimation more than k times. Finally, we combine the $i+1$ distributions into one using convolution. Thus, we assume that all sub-paths that

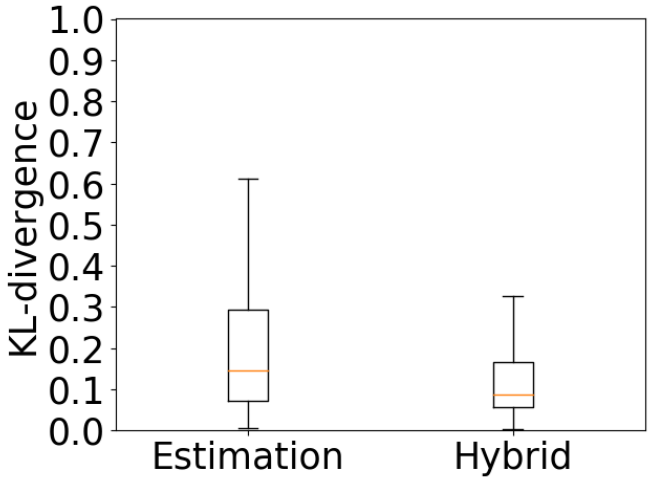


Figure 8: Boxplot of KL divergence between results of estimation, the hybrid method, and the ground truth. Based on more than 500 samples, with $n = 10$ and bucket width $b = 2$

Algorithm 1: Path cost

Input:
Path $P = \langle e_1, .., e_k \rangle$; Integer k;
Output:
Cost distribution of path P;

```
For every two consecutive edges in P, use classifier to mark shared vertex n.conv or n.est;
D ← Distribution of e_1;
S_P ← Set of distributions to be convoluted;
i ← 2; len ← 1;
while i < |P| do
    if len = k or e.s.conv then
        Insert D into S_P;
        len ← 1;
        D ← e.cost;
    else if e.s.est then
        D ← NN(D, e);
        len ← len + 1;
    i ← i + 1;
result ← Convolve all distributions in S_P into one;
return result;
```

the distributions represent are independent, which is justified by the classifier. The assumption may not hold when $len = k$, but estimation becomes inaccurate after repeating it k times.

IV. Routing with Hybrid Learning

A. 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. The algorithm expands outwards from s in a breadth-first manner, maintaining a minimum-cost priority queue containing all unexplored vertices organized according to the costs of reaching them via the explored network. The algorithm repeatedly considers the lowest-cost vertex from the queue, establishing a final cost for that vertex during extraction. The cost is guaranteed to be minimal, as it is impossible to reach the vertex from any other unexplored vertex via a lower-cost path. This guarantee holds specifically because weights are non-negative. The algorithm terminates when the destination vertex d has been extracted from the queue. As an example, examine Figure 10 and assume it is cheaper to traverse P_1 than P_2. As traversal costs are calculated as sums, we can guarantee that $P_1' = P_1 + e$ has lower cost than $P_2' = P_2 + e$.

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 notion of lowest-cost path can be defined as the path with a stochastically non-dominated distribution among all paths with the same source and destination. This implies that 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_{D_1} and CDF_{D_2}, we say that D_1 dominates D_2 if:

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

Here, 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.

Conversely, if distributions D_1 and D_2 are identical or if the inverse statement,

$$\exists x\ (CDF_{D_1}(x) > CDF_{D_2}(x)) \wedge \exists x\ (CDF_{D_1}(x) < CDF_{D_2}(x))$$

is true, we say that the two distributions do not dominate each other, i.e., they are pair-wise non-dominated. Given the above scenario of finding a non-dominated distribution, we could either pick one distribution to continue with and prune the other, or we could branch out and continue with both.

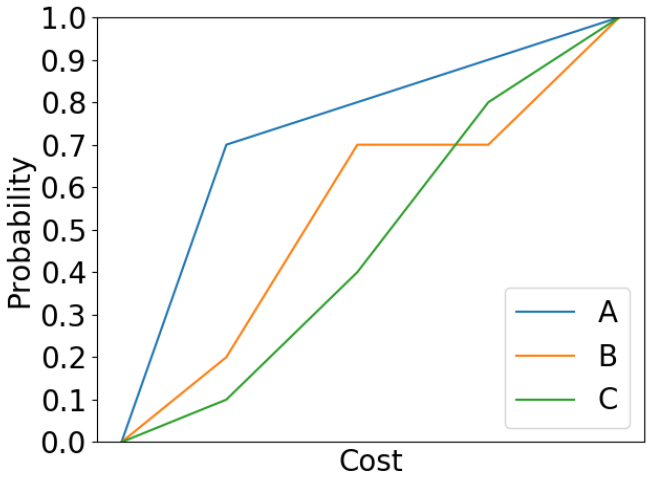


Figure 9: Stochastic dominance example

As an example, consider the three CDFs depicted in Figure 9. Each CDF represents a unique path between the same source and destination. As can be seen, all three distributions share the same minimum and maximum costs. Following the aforementioned definition of stochastic dominance, we can determine pairwise dominance relationships by examining the CDFs in the figure. Distribution A dominates B, as there is no x for which $CDF_A(x) < CDF_B(x)$, whereas it is always the case that $CDF_A(x) \geq CDF_B(x)$. A dominates C in a similar manner. Furthermore, B does not dominate C and vice versa. This is evident, as the lines representing the two distributions intersect, meaning that we have values of x for which $CDF_B(x) < CDF_C(x)$ and values for x for which $CDF_B(x) > CDF_C(x)$. Thus, given three paths with CDFs A, B, and C, we can prune B and C as the cumulative probability of any cost never exceeds that of A.

Given our setting with a routing time budget t in an uncertain network suggests that we simply pick and proceed with the distribution with the highest probability of cost t. However, that is not possible. To see why, assume a time budget t and consider paths P_1 and P_2 in Figure 10. Furthermore assume that $Prob(CDF_{P_1}, t) > Prob(CDF_{P_2}, t)$. Even though P_1 is better than P_2, we cannot simply disregard P_2 because it can happen that $Prob(CDF'_{P_1}, t) < Prob(CDF'_{P_2}, t)$. Consequently, we cannot use a budget to prune paths reaching intermediate vertices [13].

Next, when using neural networks to estimate path cost distributions, we may not be able to perform pruning based on stochastic dominance. First, the output estimate may not stochastically dominate either of the input distributions. When this happens, the result distribution has a lower expected value than those of the input distributions have. If this occurs too often, it is a problem, as we could in a worst-case scenario repeatedly expand a path while maintaining a non-increasing cost. To avoid this, we ensure that the minimum cost of the output distribution is bounded by the sum of the minimum costs of the two inputs. Furthermore, we cannot guarantee that the dominance relationship between two paths also holds for the paths when they are extended by the same edge. For instance, consider Figure 10 where paths P_1 and P_2 reach vertex v_k, and assume that the CDF of P_1 stochastically dominates that of P_2. Given $P'_1 = NN(P_1, e)$ and $P'_2 = NN(P_2, e)$, in a traditional setting, it would hold that the CDF of P'_1 stochastically dominates that of P'_2. However, with hybrid cost computation, we cannot know what relationship the cost distributions of $P'_1 = NN(P_1, e)$ and $P'_2 = NN(P_2, e)$ have, as a neural network is a complex, non-linear function. This breaks the subpath optimality property, which causes issues for routing.

Thus when using Dijkstra's algorithm, and replacing convolution with estimation, we find that pruning cannot be done on all vertices since the subpath optimality property does not hold. As a consequence, we cannot directly apply Dijkstra's algorithm in combination with 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

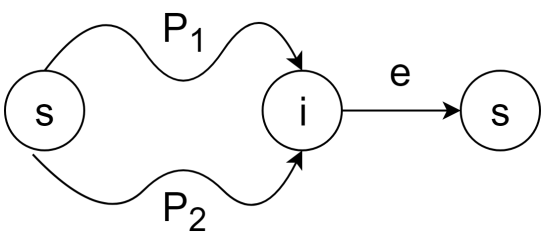


Figure 10: Two paths being extended by the same edge

organized on the cost expectation. An example of such a brute force search is shown in Algorithm 2. This approach is naturally very inefficient, as the entire search space is explored before a path is returned.

Algorithm 2: 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 ← priority queue of paths sorted on expected cost;
2:  Insert all outgoing edges from s into PQ;
3:  result ← set of non-dominated paths to be returned;
4:  while PQ is not empty do
5:      P ← extract-min from PQ
6:      for all e ∈ outgoing neighbors of P.d do
7:          P' ← ⟨P, e⟩, calculate cost using Algorithm 1;
8:          if P'.d = d then
9:              result ← argmax_{x∈{P',result}} prob(x, t);
10:         else
11:             if P' is a simple path then
12:                 insert P' into PQ;
13: Return result;
```

Algorithm 2 returns the best path given budget t between vertices s and d. It iteratively expands the search outwards 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 $result$ represents the best current path found between s and d, and whenever a new path between s and d is found, we compare it to $result$. The brute-force approach utilizes Algorithm 1 to construct the traversal cost of a path. However, this approach introduces overhead because the traversal cost is fully recomputed each time a path is expanded.

Being brute force, Algorithm 2 is inefficient due to two main reasons. First, in line 7, whenever we extend a path P with an edge e to obtain a new path $P' = \langle P, e \rangle$, we call Algorithm 1 to compute the cost distribution of P'. Algorithm 1 computes the cost distribution from scratch, meaning that it does not reuse the cost distribution of path P that has already been computed. Second, no pruning is applied, and the search space is very large.

To improve the efficiency of the brute force algorithm, we consider the incremental property of the hybrid learning

approach that will enable reuse of already computed cost distributions; and we propose a hybrid routing algorithm that allows pruning at selected independent vertices.

B. Incremental Property

We do not wish to recompute the entire cost distribution each time we add an edge to a path. Instead, we present a method that supports incremental path extension by caching and reusing different distribution elements of a path. Algorithm 3 details this process. Note that Algorithm 1 is designed for computing the cost distribution of a path given the distributions of all the edges in the path, whereas Algorithm 3 is designed for computing the cost distribution of a path $P' = \langle P, e \rangle$ using the distributions of P and e.

In Algorithm 3, 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 the 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 ML estimation and then convolve D_1 and D_2.

Algorithm 3: Incrementally Build Path Cost

Input:
Path $P = \langle e_1, .., e_k \rangle$; Edge e_n;
Output:
Path $P' = \langle e_1, .., e_k, e_n \rangle$ with associated traversal cost;

```
op ← Determine with classifier if convolution or
     estimation should be used for calculating the cost of
     ⟨e_k, e_n⟩;
P' ← ⟨P, e_n⟩;
if op = convolution then
    P'.D_1 ← P.D_FC ⊕ e_n.cost;
    P'.D_FC ← P'.D_1;
    P'.D_2 ← empty;
else
    P'.D_1 ← P.D_1;
    if P.D_2 is empty then
        P'.D_2 ← e_n.cost;
    else
        P'.D_2 ← NN(P.D_2, e_n.cost);
    if P.D_1 is empty then
        P'.D_FC ← P'.D_2
    else
        P'.D_FC ← P'.D_1 ⊕ P'.D_2;
if P'.D_2 represents the cost of a path of length k then
    P'.D_1 ← P'.D_FC;
    P'.D_2 ← empty;
return P';
```

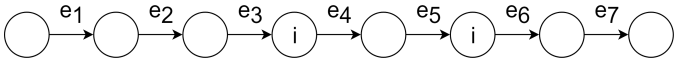


Figure 11: Example graph for path expansion with $k = 3$

Table V: Cost elements for paths

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

Figure 11 shows a path with 7 edges, where the independent vertices are marked with "i", e.g., the vertex between e_3 and e_4. Starting from e_1, we iteratively extend the path with an additional edge. Table V shows the different cost elements for each path throughout the expansion. Distribution D_2 always represents the estimation 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 estimated sum of the costs of e_1 and e_2. In this example we have $k = 3$, 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 of three dependent vertices, and the result of the estimation is therefore moved into 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 unnecessary repeats 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 cost estimation on the distributions given by e_5 and $P_4.D_2$, and then convolving the result of the estimation with the distribution stored in $P_4.D_1$. Paths P_6 and P_7 follow the same procedure, representing the total path cost by convolving D_1 and D_2.

C. Hybrid Routing

We integrate the incremental path cost construction with more sophisticated hybrid search. The hybrid search, detailed in Algorithm 4, uses several pruning techniques and can be considered an advanced version of the brute-force approach. 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, which is used 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 finally (d) stochastic dominance pruning on all fully independent vertices, i.e., vertices with all combinations of in-edges and out-edges being classified as independent.

We initiate the 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 the edge's distribution. Then we identify a sub-graph $G' = (V', E')$, where V' represents all reachable vertices from source s within time t and E' include 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.

Second, to enable A*-like search, we need to estimate an optimistic cost from each vertex to the destination vertex d. 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. We also determine the fastest path when using minimum travel costs. Third, we perform a new Dijkstra search on G', where each edge is annotated with the maximum travel cost. We identify the fastest path using the maximum travel costs.

The algorithm explores paths using a priority queue PQ. The queue is sorted on (a, b), where a is the optimistic probability of arriving at d and b is the optimistic expected travel time to d. By utilizing both a and b, we increase the likelihood of finding promising paths between s and d early. 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 if we continue along the path to d. We use D_1 and not D_{FC} because the nature the estimation may lead to a path expansion yielding 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 return statement in line 17, 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 a and b. Figure 12 illustrates this shifting method when $v.min = 6$. The shifted distribution represents the most optimistic distribution if we continue from path P to reach the destination d.

Recall that the two final searches determine paths representing the fastest optimistic and pessimistic paths between s and d in a deterministic setting. The best of these two paths 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 a new pivot, in lines 18–20. The pivot path is guaranteed to have a non-zero probability of reaching d within t time budget and is used for

Algorithm 4: Hybrid Search

Input:
Graph $G = (V, E, W)$; Time budget t;
Source s and destination d;

Output:
Path with maximum probability of arriving at d by t;

1: $H_{PS} \leftarrow$ hash map of (vertex, path set);
2: $V' \leftarrow$ all vertices reachable from s within t time using the minimum traversal time of each edge;
3: One-to-all Dijkstra search from d using minimum travel costs, annotating all vertices with the minimum cost of reaching d;
4: One-to-all Dijkstra search from d using maximum travel costs, annotating all vertices with the maximum cost of reaching d;
5: **if** V' does not contain d **then**
6: Return;
7: $P_a \leftarrow$ using Algorithm 1, build the fastest optimistic path that is found using minimum traversal time;
8: $P_b \leftarrow$ using Algorithm 1, build the fastest pessimistic path that is found using maximum traversal time;
9: $result \leftarrow \text{argmax}_{x \in \{P_a, P_b\}} Prob(x, t)$;
10: **if** $Prob(result, t) = 1.0$ or $P_a.edges = P_b.edges$ **then**
11: Return $result$;
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: $P \leftarrow PQ.ExtractMin()$;
16: **if** $Prob(P.D_1 + P.d.min, t) < Prob(result, t)$ **then**
17: Return $result$;
18: **if** $P.d = d$ **then**
19: $result \leftarrow \text{argmax}_{x \in \{result, P\}} Prob(x, t)$;
20: Continue;
21: **else if** $P.d$ is a *fully-independent* vertex **then**
22: Insert P into $H_{PS}[P.d]$ and update such that no dominance occurs within $H_{PS}[P.d]$;
23: Remove all paths P_m in PQ that go through $P.d$, where sub-path $P'_m = \langle s, \ldots, P.d \rangle \notin H_{PS}[P.d]$;
24: **if** $P \notin H_{PS}[P.d]$ **then**
25: Continue;
26: **for all** $e \in$ Outgoing edges of $P.d$ where $e.d \in V'$ **do**
27: $P_k \leftarrow \langle P, e \rangle$, calculate cost;
28: **if** P_k is not simple **then**
29: Continue; // Paths with loops are never best
30: $L \leftarrow P_k.D_1$ shifted $e.d.min$ to the right; // Most optimistic probability of reaching d within t;
31: **if** $Prob(L, t) \geq Prob(result, t)$ **then**
32: Continue;
33: Insert P_k into PQ with queue cost $(Prob(L, t), expectation(P_k) + e.d.min)$;
34: Return $result$;

comparison with all 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 thanks to 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 that has a higher probability of arriving within t than the pivot path. If the probability is the same or exceeds that of the pivot, we proceed with routing using P.

Finally, we perform stochastic dominance based pruning on all fully independent vertices, as seen on lines 21–25. A vertex is fully independent if the vertex is classified as independent for all combinations of in-edges and out-edges of the vertex. This means that 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 use stochastic dominance to perform pruning.

We use a hash map H_{PS} 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 pre-discovered non-dominated paths ending at n to determine whether it is necessary to expand P any further.

Note that the fully independent vertices are different from the independent vertices used in Algorithm 3. In Algorithm 3, 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 on two distinct occasions. During routing, we iteratively add an edge to an existing path (line 27), and during the initial stage, we build the costs for the fastest optimistic and pessimistic paths, in lines 7–8. We distinguish these two occasions because of the different input used; in the first cases, an entire path is given, and we therefore use Algorithm 1 to construct a cost distribution, whereas we use Algorithm 3 in the path expansion process where we extend paths with a single edge.

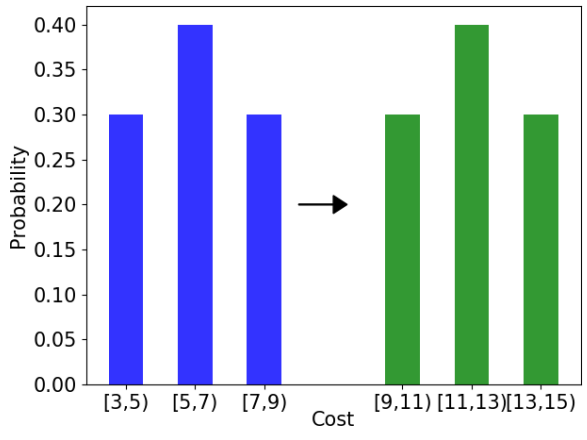


Figure 12: Shifting a distribution (blue) $v.min = 6$ to the right (green)

Extension for Anytime Query Processing. Algorithm 4 performs well on average, but has a mean run-time that is dominated by few "outlier" queries being disproportionally slow. This is among other things due to being unable to limit the search space further than V', which may be very large when the source and destination are far apart.

To control the run-time, we propose an anytime extension to Algorithm 4 that limits the time spent searching. With this approach, we give an acceptable maximum run-time t_a as an additional input, and the algorithm returns the pivot path if search has not terminated when it has spent t_a seconds searching for a path. The pivot path is the best path found at any time during the execution of the algorithm — the closer the algorithm is to termination, the more likely it is that the pivot path is different from P_a or P_b, assuming that the best path is different from these.

Furthermore, we modify the priority queue to use a different sorting for paths: we now use the A* cost estimate on the final vertex of a path. The intuition is that we wish to find as many promising candidate paths as possible before termination, and sorting on the A* cost estimation means that paths closer to the destination have a larger priority than paths farther away.

Observe that this extension breaks the guarantee of returning the path with the largest probability of arriving by t. Instead, Algorithm 4 now returns a path that is at least as good as $\hat{P}$, which is the best path using deterministic minimum edge costs. In the next section we assess the quality of the paths returned by the extended algorithm.

V. Empirical Study

A. Experimental Setup

Road network and GPS Data. We use an undirected graph that represents the Danish road network, extracted from OpenStreetMap[1]. The graph consists of 667,950 vertices and 1,647,724 edges. We utilize a GPS data set consisting of ca 180 million GPS records, covering 167,520 edges of the graph. All measurements are rounded to the nearest second. Further, we disregard measurements that are rounded to 0, and we disregard trajectories representing road segment traversal speeds exceeding 110% of the speed limit. This ensures that no roads exist which are free to traverse, while also making sure we avoid producing paths that are best only if a driver exceeds the speed limit.

Uncertain Road network. We instantiate the cost function $C : E \rightarrow D$ in G as follows: First, if an edge is covered by GPS records, we instantiate the travel time distribution of the edge using the records. Second, if an edge is not covered by GPS records, we derive a travel time t_d based on the length and speed limit of the edge. Then, we generate a triangular distribution centered around $t_d \cdot 1.2$ with lower bound t_d and upper bound $t_d \cdot 1.4$. The intuition is that drivers

[1] http://www.openstreetmap.org

may not always drive as fast as the speed limit due to traffic and often spends more than t_d. We use this approach to ensure uncertainty across all edges. A triangular distribution is similar to a Gaussian distribution, the main difference being that it accepts strict upper and lower bounds as parameters, ensuring no samples have values outside the bounds. Further, we center the triangular distribution around $t_d \cdot 1.2$ to reduce the likelihood of performing routing with edges having no trajectory data. Further, we always use convolution on edges with triangular distributions.

Time Budgets. The budgets in probabilistic budget routing queries have a significant impact on routing efficiency. Selecting a very large budget enables Algorithm 4 to short-circuit and return one of the two paths found in the deterministic search, since each path has probability 1.0 of arriving at d by t. Conversely, a very small budget decreases the size of the search space given by V', which in turn improves query efficiency. However, if the budget is too small, no path is able to reach d within t. Thus, in order to assess the impact of different budgets, we need to select time budgets carefully.

We proceed to describe how we choose time budgets. Given a source and destination pair, we employ Algorithm 4, but terminate after line 7, which returns path P_a, the fastest optimistic path when all edges are annotated with minimum travel times. Next, based on the travel time cost distribution, we choose three time budgets b_1, b_2, and b_3, such that the probabilities that the path P_a has travel time smaller than b_1, b_2, and b_3 are 25%, 50%, and 75%, respectively. In other words, we choose the 25%, 50%, and 75% quantiles of the distribution of P_a as the budgets.

Queries. To speed up the routing, we pre-compute dependence relations in the graph. We apply a classifier to each pair of adjacent edges to determine whether they are cost dependent. Edges with no trajectory coverage are assumed to be independent in all associated relations. Roughly 10% of all edge pairs are cost dependent.

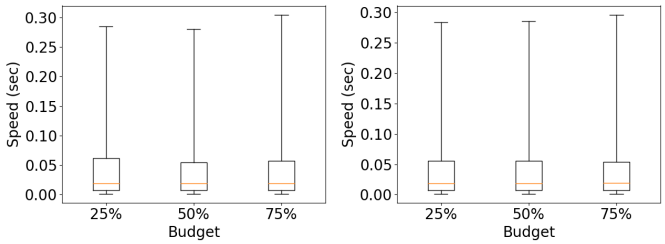
To easily distinguish between the results of Algorithm 4 with and without the anytime extension, we denote the returned path as P_x, where x is the time limit. Thus, P_∞ denotes the path returned by the algorithm without anytime extension. When using the anytime extension, we focus on P_1, P_5, and P_{10}, i.e., the paths returned with a 1, 5, and 10 second time limit.

We focus on probabilistic budget routing in intra-city settings as the traffic uncertainty inside cities is higher. The path choices for inter-city travel are often limited because they tend to use highways. Thus, we generate source and destinations pairs inside cities based on three distance categories: $[0, 1), [1, 5)$, and $[5, 10)$. For each category we generate 100 source-destination pairs that we use as input to Hybrid Search to examine the influence of different budget values and varying values of k.

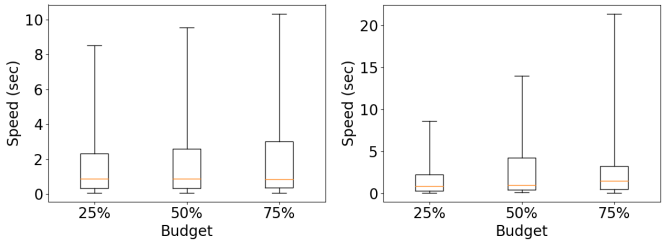
Implementation. All algorithms are written in Python version 3.7. The experiments are conducted in a single process on a machine running Windows 10 with an 8-core Intel 8900K 4.2 GHz CPU with 32GB DDR4 main memory and 4TB secondary memory.

B. Experimental Results

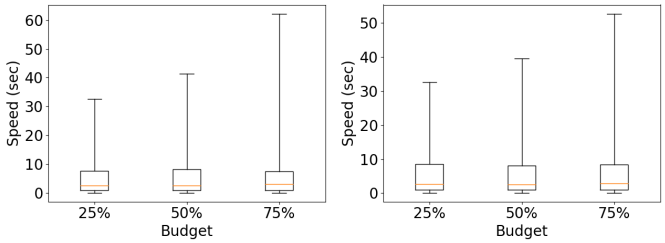
1) Routing Efficiency: For routing efficiency, we consider Algorithm 4 without the anytime extension, i.e., P_∞.



(a) Efficiency, $[0, 1)$, $k = 2$ (b) Efficiency, $[0, 1)$, $k = 4$



(c) Efficiency, $[1, 5)$, $k = 2$ (d) Efficiency, $[1, 5)$, $k = 4$



(e) Efficiency, $[5, 10)$, $k = 2$ (f) Efficiency, $[5, 10)$, $k = 4$

Figure 13: Boxplots visualizing the speed of Algorithm 4. Whiskers represent 5th percentile and 95-th percentile

Figures 13(a–f) report on the runtimes of Algorithm 4 using different query distance categories, different values of k, and different budgets. We omit figures for $k = 3$ to save space. In general, short queries terminate very fast no matter the configuration. Examining the results for longer queries reveals that larger budgets lead to an increased variance in execution time. In contrast, varying k while maintaining the same budget has a negligible effect on the execution time; in the $[5, 10)$ query category, the mean varies by at most one second when varying k.

For all categories, the mean runtime 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 river, we may need to take a detour to cross a bridge. Second, varying numbers of measurements on edges may yield loose path cost

bounds. This happens in cases with many traversals on edges. Here, measurements of traversals at very low speeds give some edges a small probability of having a very high traversal cost. This yields a larger search space for Algorithm 4. Third, the nature of estimation means that an expanded path P may be better than the pre-path P^{-1}. When this occurs, path pruning occurs less frequently than when using convolution. Fourth, the budget size has a direct impact on the search space. This is reflected in the figures, as can be seen by looking at the worst case for different budgets — using a smaller budget can significantly reduce the worst-case execution time, and thereby also the mean.

Table VI: Probability of returning a path P different from $\hat{P}$

[0, 1)	25%			50 %			75 %		
Path	k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
P_∞	12%	12%	12%	12 %	13%	13%	14%	15%	14%

[1, 5)	25%			50 %			75 %		
Path	k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
P_∞	49%	51%	54%	50%	52%	54%	53%	55%	57%
P_1	47%	50%	52%	48%	50%	52%	50%	53%	54%
P_5	49%	51%	54%	50%	52%	54%	53%	55%	57%
P_{10}	49%	51%	54%	50%	52%	54%	53 %	55%	57%

[5,10)	25%			50 %			75 %		
Path	k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
P_∞	59%	56%	56%	60%	64%	58 %	65%	65%	60%
P_1	54%	48%	46%	56%	58%	50%	62%	61%	54%
P_5	59%	55%	54%	59%	62%	57%	63%	63%	59%
P_{10}	59%	55%	56%	60%	62%	58%	64%	64%	59%

Table VII: Average probability of returning a path P different from $\hat{P}$

	P_∞	P_1	P_5	P_{10}
$[0, 1)$	13%	13%	13%	13%
$[1, 5)$	53%	51%	53%	53%
$[5, 10)$	60%	54%	59%	60%

2) Routing Quality: To assess the quality of the returned paths, we first examine the likelihood of P_∞ being different from $\hat{P}$, where $\hat{P}$ is the path given by line 9 in Algorithm 4. This is of interest because if $P_\infty \neq \hat{P}$, a strong cost dependence exists in the search space; otherwise, the two paths P_∞ and $\hat{P}$ should be identical. If there is no strong cost dependence, there is no need to use Hybrid Routing. In addition, it is possible to pre-compute whether we should use Hybrid Routing for a given source and destination pair by determining whether $P_\infty = \hat{P}$. Observe that $\hat{P}$ may be the best path no matter which type of routing algorithm we use. Furthermore, for each query, it is always the case that if $P_x \neq \hat{P}$ then $P_\infty \neq \hat{P}$, and if $P_\infty = \hat{P}$ then $P_x = \hat{P}$, where P_x can be, e.g., P_1, P_5, or P_{10}. As a consequence, no P_x can have a larger percentage of being different from $\hat{P}$ than P_∞.

Table VI shows the percentage of times P_∞, P_1, P_5, and P_{10} are different from $\hat{P}$ for each query category, and Table VII provides a summarized version to better see the trend. All short queries finish within a second. Hence, P_1, P_5, and P_{10} are equal to P_∞. Further, for short queries, $\hat{P}$ has a very high likelihood of being the best path. This is likely because most paths are so short ($|P|$<5) that the uncertainty and dependence do not matter.

However, for longer queries, the uncertainty and dependencies make a difference. In the medium distance category, we obtain a path different from $\hat{P}$ some 50% of the time. Here, P_1 and P_∞ are very similar, with a difference of 2%, suggesting that it often is not worth spending more than one second on these queries. In the long distance category, we find that $P_\infty \neq \hat{P}$ 60% of the time on average. Also, P_{10} is very close to P_∞, although P_{10} is limited to ten seconds, while P_∞ can spend many times that in the worst case as seen in Figures 13(e–f).

Next, we examine the set of returned paths where $P_x \neq \hat{P}$ to assess the differences in terms of traversed vertices and probability of arrival within the time budget.

Table VIII shows the difference in probability of arrival within t between P_x and $\hat{P}$. This probability always larger than 0.0 and smaller or equal to 1.0. Interestingly, the average difference is very large, suggesting that when there is a strong cost dependency, paths may exist that are significantly better than $\hat{P}$. Moving from left to right in the tables, we observe that the average difference diminishes. This is due to the probability of $\hat{P}$ being at least as large as the budget percentage, e.g., 25%. Thus, there is more leeway to improve the probability with a low budget.

Examining P_x vs. P_∞, we find that between five to ten seconds is an acceptable limit for queries in $[1, 5)$ and that queries in $[5, 10)$ need at least ten seconds. However, with a large budget, it is also more acceptable to decrease the time limit for P_x. For example, for 75% in $[5, 10)$, we obtain the same result with P_5 as with P_{10}, suggesting that time limits should be stated relative to budget sizes. Conversely, there is no obvious correlation between k and the likelihood of $P_\infty = P_x$. From Tables VI and VIII, we learn that we sometimes find a path P_x different from both $\hat{P}$ and P_∞. For example, in the $[1, 5)$ category for P_5 with $k = 2$ and 25%, we have a lower average probability difference than that of P_∞; yet, according to Table VI, the same number of paths are different from $\hat{P}$ with this configuration. Thus, cases exist where the algorithm finds a result that is better than $\hat{P}$, but is not the best. Still, the average difference in probability is 5%, which can be considered negligible.

Next, we consider the similarity between P_∞ and $\hat{P}$. Specifically, we calculate the Jaccard similarity between their vertex sets, having excluded the source and destination vertices that always occur in both sets. The Jaccard similarity is defined as $JS = \frac{|A \cap B|}{|A \cup B|}$. A similarity score of 1.0 means the two paths are identical, whereas a similarity score of 0.0 means the two

Table VIII: Average difference of the probability of arrival by t between P_x and $\hat{P}$ when $P_x \neq \hat{P}$

[0, 1)	25%			50 %			75 %		
Path	k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
P_∞	0.38	0.36	0.33	0.26	0.22	0.20	0.14	0.11	0.10

[1, 5)	25%			50 %			75 %		
Path	k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
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)	25%			50 %			75 %		
Path	k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
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 IX: Similarity between P_∞ and $\hat{P}$ when $P_\infty \neq \hat{P}$

P_∞	25%			50 %			75 %		
Length	k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
$[0, 1)$	0.47	0.40	0.45	0.43	0.37	0.42	0.39	0.34	0.41
$[1.5)$	0.47	0.46	0.49	0.47	0.43	0.47	0.46	0.44	0.46
$[5, 10)$	0.48	0.47	0.45	0.48	0.50	0.48	0.47	0.48	0.48

paths are disjoint and thus share no vertices. Table IX reports the average Jaccard similarities. The table shows that hybrid paths on average are very different from $\hat{P}$, scoring 0.50 when they are most similar. In other words, paths follow, on average a route so different that half of the vertices covered by the path are different from the covered vertices in $\hat{P}$. The average number of vertices covered by P_∞ for $[0, 1)$ is 12.7, while for $[1, 5)$, it is 40.7, and for $[5, 10)$, it is 69.3. Conversely, the average number of vertices covered by $\hat{P}$ for $[0, 1)$ is 12.7, while for $[1, 5)$, it is 41.3, and for $[5, 10)$, it is 70.5. Thus, paths have similar lengths, but cover very different sets of vertices. No matter the distance category, $P_\infty \neq \hat{P}$ implies very different paths, showing that considering cost dependence can significantly alter results, particularly for long distance queries.

Next, we examine the similarity between P_x and P_∞. We disregard $[0, 1)$ because $P_x = P_\infty$ for all categories. Table X shows that paths in $[1, 5)$ are very similar, 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 in this distance category. Similarly, using 1 second for $[5, 10)$ queries yields the same result as P_∞ in many cases, and all tested time limits yield results very similar to P_∞. Further, using 10 seconds instead of 5 seconds offers little additional benefit.

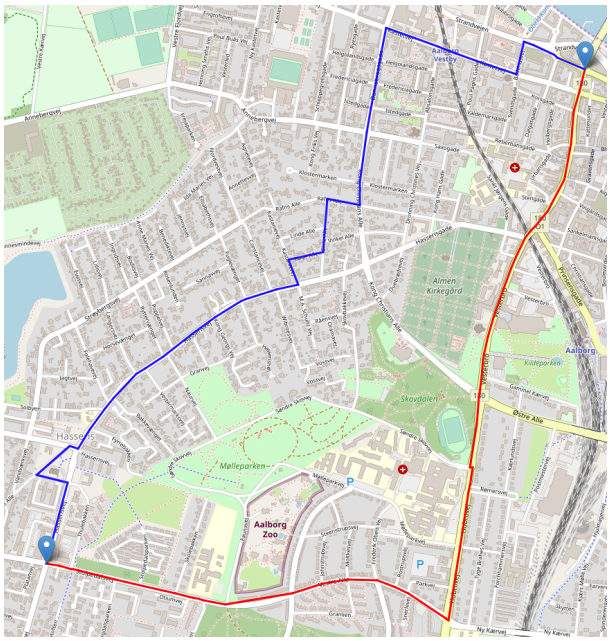


Figure 14: Real-life routes computed by $\hat{P}$ (red) and P_∞ (blue) with $k = 3$ and a 300 second budget

Figure 14 shows an example of query results computed by $\hat{P}$ and P_∞. In this example, $P_a = P_b = \hat{P}$, and P_∞ utilizes more than 50000 measurements to derive a distribution. The paths suggest that $\hat{P}$ tends to follow main roads, likely because of their larger speed limits, which reduce the minimum traversal cost. However, P_∞ is significantly different from $\hat{P}$, suggesting that main roads have traversal costs that are very uncertain with low probability of fast traversal. P_∞ passes through a residential area, which likely has lower speed limits and less traffic. Further, $\hat{P}$ passes eleven traffic lights, while P_∞ passes seven, but P_∞ requires eleven turns, whereas $\hat{P}$ requires one. This suggests that turn costs in residential areas are negligible, likely because a driver rarely has to stop for more than a few seconds, whereas traffic signals on main roads may produce more uncertainty due to a large difference in time stopped.

We conclude that using Hybrid Routing is a good idea in regions with strong cost dependence. Further, it is acceptable to limit the worst case execution time by using an anytime variant of Algorithm 4. For short queries, we suggest a time limit of one second, and for medium and long range intra-city queries, we suggest a five second time limit.

VI. Related work

Stochastic Cost Modeling. The field of path-cost modeling has been studied extensively, often with an underlying model assuming cost independence [5], [15], [17]. Some studies consider temporal dependence, i.e., traversal costs are given as a function of time, but assume no spatial cost dependence between neighboring edges if a departure time is given [2],

Table X: Jaccard similarity between P_∞ and P_x

[1, 5)	25%			50 %			75 %		
Path	k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
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)	25%			50 %			75 %		
Path	k=2	k=3	k=4	k=2	k=3	k=4	k=2	k=3	k=4
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

[16], [19]. This paper's cost model considers spatial dependence, i.e., adjacent edges may be cost dependent. Several studies integrate spatial dependence into the cost model by examining historical trajectories to reuse path costs [1], [6], but they are 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 models spatial dependence between edges [8]. 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 [11], but assumes that all pairs of adjacent edges have a known joint distribution. Further, it uses synthetically generated distributions. Although some studies [12], [20] employ histograms to represent travel cost distributions, they only consider individual road segment traversal cost and assume cost independence.

One study [4] models cost dependence as a correlation between all edges and a global hidden random variable. In contrast, we model local spatial dependence.

To the best of our knowledge, we are the first to propose a cost model that combines convolution and machine learning to approximate spatially dependent path costs more accurately.

Stochastic Routing. Compared to traditional routing algorithms [7], [9], where costs are assumed to be deterministic, stochastic routing algorithms employ uncertain weights representing road segment traversals. Existing stochastic routing algorithms often assume that edge cost distributions are independent and perform pruning based on stochastic dominance [14], [16], [19]. Many studies on routing implicitly assume that the subpath optimality property holds. However, one study examines how to ensure compliance of the property when solving the multi-criteria shortest path problem in a time-dependent graph model of public transportation. Here, the subpath optimality property does not hold because an algorithm aims to minimize several criteria to retrieve the shortest route, and it has the option to wait for a departure. That paper solves the problem by solely comparing paths with the same departure time, which ensures subpath optimality compliance [3]. We also find that subpath optimality does not always hold in the setting that we consider. Instead, we determine the cases in which it does hold, and we use subpath optimality for pruning only in those cases. Further, we consider spatial dependence, not temporal dependence, and do not allow waiting at any vertex. One study considers cost dependence while routing—it only uses stochastic dominance-based pruning if two edges are independent [18]. However, independence is assumed if no traversal exists that covers both edges in sequence. To achieve efficiency, we propose two additional pruning techniques using pivot paths and A* like optimistic costs. In addition, we propose an anytime extension that provides good results within a runtime limit, e.g., 5 seconds.

VII. 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 a routing algorithm. We conduct extensive experiments that offer insight into the efficiency and result quality achieved by the algorithm. Further, we build an anytime extension of the algorithm that limits the execution time, and we show that intra-city queries that may be slow in the worst case, can complete in reasonable time while offering high result quality.

In future work, it is of interest to consider personalized routing and to examine if edge properties by themselves are sufficient to determine distributions, such that no trajectories are required for cost estimation.

References

[1] S. Aljubayrin, B. Yang, C. S. Jensen, and R. Zhang. Finding non-dominated paths in uncertain road networks. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 15. ACM, 2016.

[2] M. Asghari, T. Emrich, U. Demiryurek, and C. Shahabi. Probabilistic estimation of link travel times in dynamic road networks. pages 1–10, 2016.

[3] A. Berger and M. Müller-Hannemann. Subpath-optimality of multi-criteria shortest paths in time-and event-dependent networks. 2009.

[4] A. Chang and E. Amir. Reachability under uncertainty. *UAI*, 2007.

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

[6] J. Dai, B. Yang, C. Guo, C. S. Jensen, and J. Hu. Path cost distribution estimation using trajectory data. *In PVLDB*, 10(3):85–96, 2016.

[7] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

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

[9] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[10] C. C. Heyde. *Central Limit Theorem*. Wiley StatsRef: Statistics Reference Online, 2014.
[11] M. Hua and J. Pei. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *EDBT*, pages 347–358, 2010.
[12] Y. Ma, B. Yang, and C. S. Jensen. Enabling time-dependent uncertain eco-weights for road networks. *Proc. GeoRich@SIGMOD*, pages 1–6, 2014.
[13] Y. M. Nie and X. Wu. Shortest path problem considering on-time arrival probability. *Transportation Research Part B: Methodological*, 43(6):597 – 613, 2009.
[14] E. Nikolova, M. Brand, and D. R. Karger. Optimal route planning under uncertainty. In *ICAPS*, pages 131–141, 2006.
[15] E. Nikolova, J. A. Kelner, M. Brand, and M. Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *European Symposium on Algorithms*, pages 552–563. Springer, 2006.
[16] M. P. Wellman, M. Ford, and K. Larson. Path planning under time-dependent uncertainty. In *UAI*, pages 532–539, 1995.
[17] A. B. Wijeratne, M. A. Turnquist, and P. B. Mirchandani. Multiobjective routing of hazardous materials in stochastic networks. *European Journal of Operational Research*, 65(1):33–43, 1993.
[18] B. Yang, J. Dai, C. Guo, C. S. Jensen, and J. Hu. PACE: a path-centric paradigm for stochastic path finding. *VLDB J.*, 27(2):153–178, 2018.
[19] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang. Stochastic skyline route planning under time-varying uncertainty. In *ICDE*, pages 136–147, 2014.
[20] J. Yuan, Y. Zheng, X. Xie, and G. Sun. T-drive: Enhancing driving directions with taxi drivers' intelligence. *In IEEE TKDE*, 25(1):220–232, 2013.