Learning to Route with Sparse Trajectory Sets

Chenjuan Guo, Bin Yang✉, Jilin Hu, Christian S. Jensen

Department of Computer Science, Aalborg University, Denmark
{cguo, byang, hujilin, csj}@cs.aau.dk

***Abstract*—Motivated by the increasing availability of vehicle trajectory data, we propose *learn-to-route*, a comprehensive trajectory-based routing solution. Specifically, we first construct a graph-like structure from trajectories as the routing infrastructure. Second, we enable trajectory-based routing given an arbitrary (source, destination) pair.**

In the first step, given a road network and a collection of trajectories, we propose a trajectory-based clustering method that identifies regions in a road network. If a pair of regions are connected by trajectories, we maintain the paths used by these trajectories and learn a routing preference for travel between the regions. As trajectories are skewed and sparse, many region pairs are not connected by trajectories. We thus transfer routing preferences from region pairs with sufficient trajectories to such region pairs and then use the transferred preferences to identify paths between the regions. In the second step, we exploit the above graph-like structure to achieve a comprehensive trajectory-based routing solution. Empirical studies with two substantial trajectory data sets offer insight into the proposed solution, indicating that it is practical. A comparison with a leading routing service offers evidence that the paper's proposal is able to enhance routing quality.

I. Introduction

Vehicular transportation is an important aspect of the daily lives of many people and is essential to many businesses as well as society as a whole [1]. As a part of the continued digitization of societal processes, more and more data is becoming available in the form of trajectories that capture the movements of vehicles [2], [3]. This data offers a foundation for improving vehicular transportation, including vehicle routing.

Traditional routing is *cost-centric* and aims at returning paths with minimal costs, e.g., distance, travel time, or fuel consumption. The cost of a path is computed from edge costs in *edge-based cost modeling* [4]–[8] or sub-path costs in *path-based cost modeling* [9]–[12]. In such routing, trajectory data is often used for annotating the edges or sub-paths with travel costs such as travel times; and routing services employ shortest path algorithms, e.g., Dijkstra's algorithm or contraction hierarchies [13], to return fastest, or simply shortest, paths. However, an existing study [14] suggests that local drivers who drive passenger vehicles follow paths that differ substantially from the paths computed using cost-centric routing and are often neither fastest nor shortest. Our paper also focuses on trajectory data that was generated from passenger vehicles.

We study a very different routing approach that relies on the availability of trajectories from local drivers. Assuming that local drivers implicitly take into account a multitude of factors, such as traffic conditions, turns, travel time, fuel consumption, road types, and traffic lights, when making routing decisions and thus know best which paths are preferable, we propose a methodology that utilizes paths found in historical trajectories to construct new paths between arbitrary (source, destination) pairs. We call this *trajectory-based* routing.

If historical trajectories show that many drivers traveling from a source s to a destination d follow a particular path, it is straightforward to recommend that path to drivers asking for directions from s to d. The big challenge now is how to benefit from historical trajectories when no historical trajectories capture paths from s to d. This is important because any set of historical trajectories is *sparse* in the sense that it is unlikely to provide paths for all s's and d's. For example, the road network of Denmark, a small country, contains some 1.6 million edges. Thus, if all edges are candidate s's and d's, a minimum of 2.6 trillion (s, d) pairs are needed. Given that the distribution of trajectories in a road network is skewed, an enormous set of trajectories (e.g., trillions for Denmark and quadrillions for Germany) would be needed before routing could be done by simply looking up paths of past trajectories for any (s, d) pair.

Figure 1 exemplifies the problem setting. The solid edges and filled vertices are covered by a set of five trajectories, while the dashed edges and unfilled vertices are not covered by any trajectories. For example, trajectory T_1 visited A and then J, X, Y, and B_3 before reaching B. If routing from A to B is requested, the path $A \rightarrow J \rightarrow X \rightarrow Y \rightarrow B_3 \rightarrow B$, as captured by trajectory T_1, can be recommended directly. The challenge is to enable routing for (s, d) pairs that are not connected by trajectories, e.g., (A_1, B_2) and (H, F).

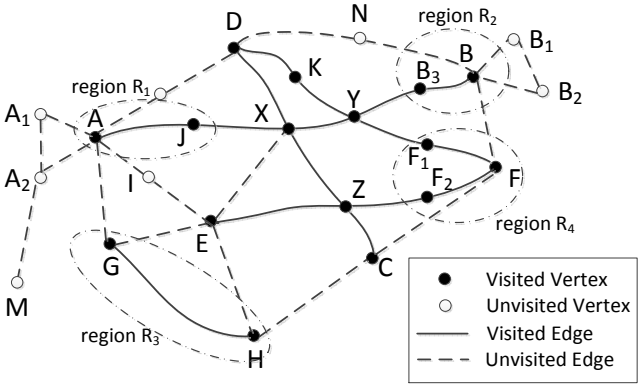


Fig. 1: Motivating Example

To enable trajectory-based routing with massive, but still sparse, sets of historical trajectories, we propose means that are able to generalize the cases where historical trajectories

can be utilized for routing. This includes *three* steps. In the first step, we cluster vertices into regions and thus map a road network graph into a *region graph*. Trajectories that originally connect vertices in the road network graph now connect regions in the region graph. This arrangement generalizes the cases where trajectories can be used for routing from being between specific vertex pairs to being between region pairs. As regions include multiple vertices, this arrangement contributes to solving the data sparseness problem.

For example, in Figure 1, A and J are clustered into region R_1, and B_3 and B are clustered into region R_2. Now, although no trajectories connect A_1 and B_2, T_1 connects regions R_1 and R_2 that are close to A_1 and B_2. Thus, the path of T_1 can be used for recommending a path from A_1 to B_2. For instance, a user may go from A_1 to A, then follow the path used by T_1 to reach B, and then go to B_2. This enables trajectory-based routing between regions connected by trajectories. However, in the region graph, some region pairs are still not connected by any trajectories, e.g., regions R_3 and R_4 in Figure 1.

In the second step, we learn routing preferences from available historical trajectories that connect some region pairs and then transfer these preferences to similar region pairs that are not connected by trajectories. Based on the transferred preferences, we identify paths for the non-covered region pairs. Note that the routing preferences are learned for different region pairs, not for different individual drivers. Assume that (R_1, R_2) is similar to (R_3, R_4), e.g., because both are from a residential area to a business district. Next, we extract a routing preference from the trajectories connecting R_1 and R_2 that explains the choice of paths from R_1 to R_2. We transfer this routing preference to driving from R_3 to R_4 and then identify paths connecting R_3 and R_4, upon which trajectory-based routing from H to F is possible.

In the third step, we provide a unified routing solution, called *learn-to-route (L2R)*, which performs path finding on the region graph, thus enabling routing between arbitrary (s, d) pairs in the original road network graph.

To the best of our knowledge, this is the first solution that learns routing preferences from historical trajectories and transfers the learned preferences to the part of a road network that is not covered by trajectories, thus supporting comprehensive trajectory-based routing for arbitrary (s, d) pairs.

The paper makes four contributions. First, it presents a trajectory-based road network clustering algorithm that produces the data foundation—the region graph. Second, it presents a general routing preference model, including an algorithm that extracts preferences from historical trajectories and an algorithm that transfers preference to similar region pairs. Third, it presents a unified routing algorithm for the region graph. Fourth, it reports on an empirical evaluation that offers insight into the proposed solution, indicating that it is capable of efficiently computing paths that match those of local drivers better than do traditional routing services.

Paper Outline: Section 2 covers related work. Section 3 covers preliminaries. Section 4 presents Step 1, region graph generation. Section 5 presents Step 2, preference learning and transfer. Section 6 presents Step 3, unified routing. Section 7 reports on empirical evaluations. Section 8 concludes.

II. Related Work

We first review studies on employing historical trajectories for **path recommendation**, considering three cases.

Case 1: Given a source and a destination, complete trajectories exist that connect the source to the destination. For example, given A and B in Figure 1, trajectory T_1 went from A to B. Then, the path of trajectory T_1 is recommended. When multiple paths exist, the path with the highest popularity is recommended, where the popularity can be defined using different strategies [15]–[17]. This is the simplest case, which is also considered in our proposal.

Case 2: Given a source and a destination, no complete trajectories exist that connect the source to the destination, but trajectories exist that can be *spliced* such that the spliced trajectories connect the source to the destination. In Figure 1, given A and F, sub-paths $A \rightarrow X$ from T_1, $X \rightarrow Z$ from T_2, and $Z \rightarrow F$ from T_3 can be spliced to form a path from A to F. Alternatively, T_1 and T_5 can also be spliced to enable a different path from A to F. To determine which spliced path is "best", absorbing Markov chains [15] and hidden Markov models [18] are employed to the probabilities that different spliced paths may occur based on historical trajectories. The spliced path with the highest probability is chosen. In contrast, we learn routing preference vectors from trajectories and apply the preference vectors to identify best paths.

Case 3: Neither complete nor spliced trajectories are able to connect a source to a destination. In the example, consider, e.g., A_1 to B_2, H to F, and M to N. Here, existing methods [15]–[18] no longer work. In this paper, the use of the proposed region graph, together with the mechanism of learning and transferring routing preferences captured by past trajectories, makes it possible to extend the situations where historical trajectories can be utilized to cover also Case 3.

Next, we review related work on **road network clustering**. Gonzalez *et al.* [19] propose a graph partition method based on prior knowledge of the road network hierarchy with l levels, which may vary from country to country. Wei *et al.* [20] propose a grid-based method for constructing regions using trajectories, where two adjacent grid cells are merged if more than τ trajectories exist that passed through them. These studies rely heavily on "appropriate" parameters, e.g., l and τ. Tuning such parameters is non-trivial. Based on recent advances in modularity based graph clustering, we propose a generic, parameter-free region generation method, where parameters such as l and τ are not needed.

Finally, we consider **learning of routing preferences** [21]–[24]. Methods [21], [22] compare the paths used by trajectories to skyline paths [7] to identify different users' dominating factors when choosing paths, e.g., travel time, fuel consumption, or distance. TRIP [23] uses the ratios between individual drivers' travel time and average travel time to model personalized travel times. A recent study from Microsoft presents an algorithm that learns driver-specific parameters for Bing

Maps' ranking function for candidate paths based on individual drivers's past trajectories [24]. However, all existing methods work only when trajectories are available. In contrast, our proposal is also able to transfer routing preferences to places without trajectories, where existing methods do not apply.

III. Preliminaries

We cover the definitions of important concepts, introduce the problem, and present a solution overview.

A **road network** is a weighted graph $\mathcal{G} = (\mathbb{V}, \mathbb{E}, \mathbb{W})$, where vertex set $\mathbb{V}$ consists of vertices representing road intersections, edge set $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$ consists of edges representing road segments, and $\mathbb{W}$ is a set of weight functions, where each function has signature $\mathbb{E} \rightarrow \mathbb{R}^+$. For specificity, we maintain four functions in $\mathbb{W}$. Functions $w_{DI}(\cdot)$, $w_{TT}(\cdot)$, $w_{FC}(\cdot)$, and $w_{RT}(\cdot)$ return the distance (*DI*), travel time (*TT*), fuel consumption (*FC*), and road type (*RT*) of the argument edge, respectively.

A **path** $P = \langle v_1, v_2, \ldots, v_a \rangle$ is a sequence of vertices where two consecutive vertices are connected by an edge.

A **trajectory** $\mathcal{T}$ is a time-ordered sequence of GPS records capturing the movement of an object, where a GPS record captures the location of the object at a time point. The time gap between two consecutive GPS records in trajectories varies, from a few seconds (a.k.a., high-frequency trajectories) to tens of seconds or a few minutes (a.k.a., low-frequency trajectories). In the experiments, we test the proposed method on both a high-frequency and a low-frequency GPS data sets. Map matching [25] is able to align a trajectory with the road-network path that the trajectory traversed. For example, the path used by trajectory $\mathcal{T}_1$ is $P_{\mathcal{T}_1} = \langle A, J, X, Y, B_3, B \rangle$.

Problem Setting. We study a new routing methodology—*trajectory-based* routing. Specifically, we study how to best utilize the paths found in trajectories to enable routing for arbitrary source and destination (s, d) pairs such that the identified paths are similar to the paths chosen by local drivers.

Spareness. The *spareness* considered in the paper means that past trajectories cannot cover paths between all possible (s, d) pairs, so simply looking up paths of past trajectories for a given (s, d) pair does not work. Although it may be possible that a substantial set of trajectories cover the roads in a road network, e.g., the 1.6 million edges in Denmark, it is almost impossible to cover all possible (s, d) pairs with paths. Having just one path for each (s, d) pair in Denmark calls for 2.6 trillion trajectories. The key challenge is to conquer data sparseness by making it possible to benefit from historical trajectories for routing from s to d when no trajectories capture paths from s to d.

Solution Overview. We propose a three-step procedure to conquer the data sparseness problem, as outlined in Figure 2.

Given a road network $\mathcal{G}$ and a set of trajectories $\mathbb{T}$, the *clustering module* employs modularity-based clustering to cluster vertices into regions, thus obtaining a region graph $\mathcal{G}_{\mathcal{R}}$. We partition the edges in a region graph into T-edges and B-edges, according to whether they are traversed or not traversed by trajectories, respectively. For each T-edge, the *preference learning module* learns a routing preference. The resulting preferences are fed into the *preference transfer module* as training data, and the *preference transfer module* transfers the preferences from T-edges to similar B-edges. Based on the learned and transferred preferences, the *routing module* recommends paths for user-specified (s, d) pairs.

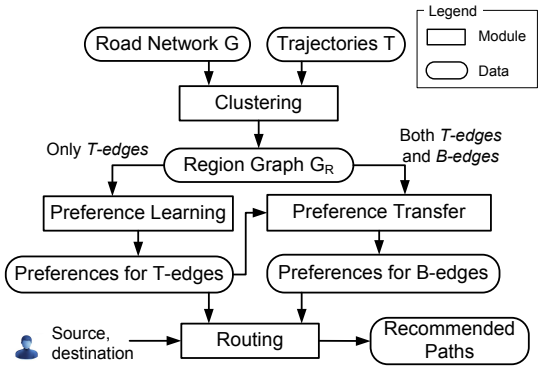


Fig. 2: Solution Overview

Scope of the paper. (1) To account for *time-dependent* traffic conditions, we construct peak and off-peak region graphs using trajectories that occurred in peak and off-peak periods, respectively. These are constructed the same way, so we disregard the distinction in the presentation. Depending on the departure time, one of the two region graphs is chosen for routing. Modeling time-dependent traffic conditions at a finer granularity and building a dynamic region graph are interesting extensions that are left for future work.

(2) *L2R* utilizes trajectories from multiple drivers to recommend paths, and thus is not a personalized routing approach. In Section VII-C, we empirically compare *L2R* with state-of-the-art personalized routing approaches. *L2R* can also be adapted to support personalized routing by only using the trajectories from specific drivers, which we also leave as future work.

IV. Building the Region Graph

We propose a trajectory-based method for *clustering* the vertices of a road network into *regions* (Section IV-A). Then, we build a *region graph* that connects pertinent regions (Section IV-B). The region graph extends the cases where trajectories can be used for recommending paths between an arbitrary pair of source and destination, thus providing a foundation for the final routing module.

A. Clustering Vertices to Regions

A **region** is a set of homogenous vertices where the homogeneity is defined based on two properties that are used in urban planning [26], [27]: (i) the numbers of trajectories associated with the vertices in a region are similar [26]; (ii) the edges connecting the vertices have the same road type [27]. The intuition is as follows. A region with vertices connected by edges of residential-road type may capture a residential area; and by taking into account the number of trajectories associated with the vertices, we can distinguish a residential area in the city from one in a suburb area because the former has more trajectories.

Consider Figure 3, where the label x:y on an edge indicates that x trajectories occurred on the edge and that the road type of the edge is y. For example, 100 trajectories occurred on edge (D, X), a type 1 road. According to the above two properties, vertices D, K, X, and Y can be regarded as a region because they have more trajectories than the other vertices and are connected by road type 1 edges. Similarly, vertices F, F_1, and F_2 can be regarded as a region.

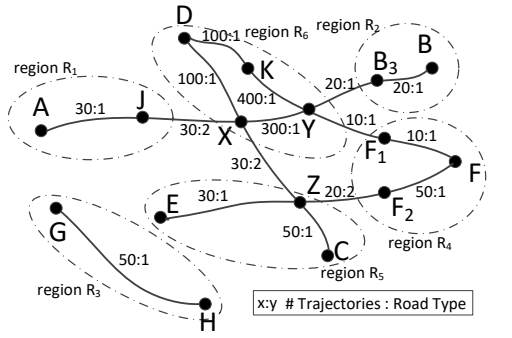


Fig. 3: An Example of Regions

Based on the two properties, we propose a modularity-based method that clusters vertices connected by the same road types into regions. The setting is a *trajectory graph* that consists of vertices and edges that are traversed by trajectories. Figure 3 shows the trajectory graph of the road network in Figure 1. A trajectory graph may not be a connected graph.

Next, we define *popularity* values for the edges and vertices in a trajectory graph. The popularity s_{ij} of edge $e = (v_i, v_j)$ is the number of trajectories that occurred on edge e. The popularity S_i of vertex v_i is the sum of the trajectories that occurred on the edges that are incident to v_i, i.e., $S_i = \sum_j s_{ij}$. Next, we define $S = \sum_{(v_i,v_j)\in\mathbb{E}} s_{ij}$ as the sum of the popularity values of all edges in the trajectory graph.

Modularity, which is used widely in the network analysis literature [28], [29], quantifies the quality of the clusters in a graph from a global perspective. In our context, the modularity is high if the *popularity* of edges inside clusters is high and the *popularity* of edges between clusters is low, which is desired by property (i) of regions.

We define *modularity gain* [28]–[30] $\Delta Q_{v_i v_j}$ to quantify the benefit of merging vertices v_i and v_j into a cluster:

$$\Delta Q_{v_i v_j} = \begin{cases} \frac{s_{ij}}{S} - \frac{S_i \cdot S_j}{S^2} & \text{if } v_i, v_j \text{ are connected by an edge;} \\ 0 & \text{otherwise.} \end{cases}$$

It has been shown that if merging two vertices v_i and v_j gives a non-positive modularity gain, the two vertices should not be merged [30]. If the modularity gain is positive, vertices v_i and v_j are merged into an *aggregate vertex* with a popularity that equals the sum of the popularity of the v_i and v_j, i.e., $S_i + S_j$.

To take into account property (ii) of regions, i.e., the road type constraint, we also associate a *road type* attribute with an aggregate vertex that records the road type of edge (v_i, v_j).

We proceed to propose a hierarchical clustering method that follows a bottom-up, agglomerative clustering strategy. In the beginning, each vertex is treated as a cluster. The method keeps merging clusters into larger clusters until no more clusters can be merged. In particular, the method merges a vertex v_k with the highest popularity, regardless of whether it is an aggregate or an ordinary vertex, with its adjacent vertices if the merging gives a positive modularity gain and only involves edges with the same road type. If v_k has no such adjacent vertices, v_k forms a region. Document [31] offers algorithmic details.

During the clustering, we need not control manually the size of clusters, as a cluster "ends" automatically when merging it with the neighbors gives non-positive modularity gains or they have different road types. This prevents naturally clusters of extremely large sizes. In addition, we maintain paths used by trajectories inside regions (see "inner-region paths" in Section IV-B). This design is useful when the source and destination in a routing request is inside a region, which is common for large regions.

Based on the above, we are able to form regions in a trajectory graph where both properties (i) and (ii) are satisfied. For example, the dashed circles in Figure 3 indicate regions. The popularity of edges in region R_6 is high, while the popularity of the edge between regions R_2 and R_6 is low; region R_6 has road type 1 edges, while the edge between regions R_6 and R_1 have road type 2.

B. Region Graph

We build a region graph $\mathcal{G}_\mathcal{R} = (\mathbb{V}_R, \mathbb{E}_R)$ based on the obtained regions, which serves as a foundation for routing. The region graph can be regarded as a backbone of the road network graph. To distinguish it from the road network graph, we call a vertex in the region graph *region vertex* and an edge in the region graph *region edge*. In particular, a region vertex $R_i \in \mathbb{V}_R$ represents a region. We proceed to show how to construct region edges by connecting region vertices, using the combination of two different strategies.

Constructing region edges from trajectories: Having identified regions, trajectories that originally connected vertices in the road network are now utilized to connect regions. If a trajectory exists that went through a vertex in region R_i and a vertex in region R_j, we construct a region edge (R_i, R_j). Note that a trajectory may produce more than one region edge. In particular, if a trajectory went through vertices in m regions, up to $\frac{m\cdot(m-1)}{2}$ region edges can be constructed. For example, in Figure 3, trajectory $\mathcal{T}_1$ went through vertices in R_1, R_6, and R_2, and we are able to construct region edges (R_1, R_6), (R_1, R_2), and (R_6, R_2), as shown in Figure 4(a).

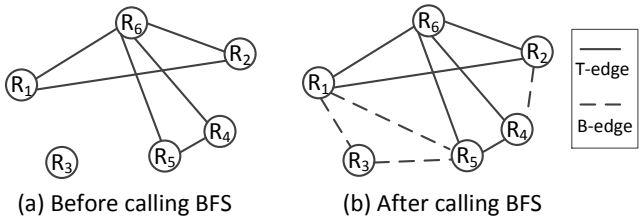


Fig. 4: Region Graph

Each region edge (R_i, R_j) is associated with a set $\mathbb{P}_{ij}$ of paths, where each path $P = \langle v_a, \cdots, v_b \rangle$ in $\mathbb{P}_{ij}$ was traversed

by at least a trajectory that left R_i at vertex v_a and entered R_j at vertex v_b. A vertex at which a trajectory $\mathcal{T}_k$ enters or leaves a region is called a *transfer center*, e.g., v_a and v_b.

For example, region edge (R_1, R_6) is associated with path $\langle J, X\rangle$ because trajectory T_1 left R_1 at vertex J and entered R_6 at vertex X, and thus J and X are transfer centers. Similarly, region edge (R_1, R_2) is associated with path $\langle J, X, Y, B_3\rangle$, where $J \in R_1$ and $B_3 \in R_2$ are transfer centers; and region edge (R_6, R_2) is associated with path $\langle Y, B_3\rangle$, where $Y \in R_6$ and $B_3 \in R_2$ are transfer centers.

For each region, we also maintain *inner-region paths* based on trajectories. Specifically, given a region R_i and a trajectory $\mathcal{T}_k$, if $\mathcal{T}_k$ entered R_i at v_c and left R_i at v_d, the path $P' = \langle v_c, \cdots, v_d\rangle$ that was traversed by $\mathcal{T}_k$ in R_i is recorded as an inner-region path of R_i. For example, regions R_1 and R_3 have inner-region paths $\langle A, J\rangle$ and $\langle G, H\rangle$, respectively.

However, when only using trajectories for constructing region edges, the resulting region graph may not be a connected graph. For example, in Figure 3, region R_3 is not connected with any other regions since no trajectory went through R_3 and other regions. Thus, we get the region graph in Figure 4(a). To enable the region graph to serve as a foundation for routing, we need to ensure that the region graph is connected. To this end, we apply a breadth first search (BFS) based procedure to make the region graph connected.

To ease the following discussion, we call the region edges that are constructed from trajectories **T-edges** and the region edges that are constructed from the BFS procedure **B-edges**.

BFS construction of region edges: We consider the original road network graph $\mathcal{G}$. We conduct a BFS for each vertex u_i in a region R_i. When the search reaches a vertex u_j in a different region R_j, we stop further exploring u_j's neighbors so that the search does not enter another region R_k via R_j. If no T-edge or B-edge exists between regions R_i and R_j, we build a B-edge as their region edge. We repeat the same procedure until all vertices in region R_i are traversed. The method of obtaining specific paths for B-edges will be discussed in detail in Section V.

For instance, consider vertex G in region R_3 in Figure 3 and the original road network graph in Figure 1. A BFS starting from G visits vertices A and E. Since vertex A is in region R_1, a region edge (R_3, R_1) is constructed as a B-edge. Similarly, since vertex E is in region R_5, a region edge (R_3, R_5) is constructed as a B-edge. The same procedure is applied to the other vertex in region R_3, i.e., vertex H, but it does not produce any new B-edges. After applying the same procedure to each region, we obtain the final region graph shown in Figure 4(b).

Different from T-edges that are composed by trajectory paths, B-edges have no path information because no trajectories went through the regions connected by the B-edges. To enable routing on top of the region graph, we need to know the paths when traveling between two regions that are connected by B-edges. To this end, in Section V, we study how to learn and transfer appropriate paths for B-edges.

An alternative way to make the region graph connected is to connect every region pair, i.e., making the region graph fully connected. However, the BFS based procedure has two benefits. First, it guarantees that there are no disconnected regions. Second, it tries to connect a disconnected region to its nearby regions, which makes the region graph simple.

V. Identifying Paths for B-Edges

To enable routing using the region graph, we associate appropriate paths with all B-edges using a three-step method. First, for each T-edge, we learn a routing preference from the set of paths that are associated with the T-edge, which explains why drivers choose specific paths. Second, we quantify the similarity between T-edges and B-edges, and then we transfer routing preferences from T-edges to B-edges based on similarity. Third, we apply the transferred routing preferences to identify appropriate paths for B-edges.

A. Step 1: Learning routing preferences for T-Edges

Each T-edge (R_i, R_j) is with a set of paths $\mathbb{P}_{ij}$ (see Section IV-B) that connects region R_i to region R_j. We learn a representative routing preference vector V_{ij} for each T-edge (R_i, R_j) that explains why drivers chose the paths in $\mathbb{P}_{ij}$.

We consider two categories of features that may affect a driver's travel decisions—travel costs and road conditions. Travel cost features describe the travel costs that drivers want to minimize. Road condition features describe drivers' preferences or restrictions relating to road conditions. For example, we may consider three different travel cost features, travel time (TT), distance (DI), and fuel consumption (FC); and three road condition features, e.g., highways, residential roads, and highways and residential roads.

Based on the above, we use a 2-dimensional vector to represent a routing preference, where the so-called master dimension corresponds to travel cost features and the so-called slave dimension corresponds to road condition features. For example, vector $V = \langle$TT, Highway$\rangle$ indicates a preference for minimizing travel time and using highways.

Based on the routing preference model, we aim at identifying an appropriate preference vector for the T-edge (R_i, R_j) based on its path set $\mathbb{P}_{ij}$. Given the source and destination of a path $P_k \in \mathbb{P}_{ij}$ and a preference vector V, we are able to construct a path P_k^V based on V that connects the source and destination of P_k. If V captures the driver's preferences well, path P_k^V should largely match the actual, or ground truth, path P_k. Thus, we aim to identify a routing preference vector V^* such that the constructed paths match the paths in $\mathbb{P}_{ij}$ as much as possible. Equivalently, we aim at solving the optimization problem $V^* = \arg\max_{V\in\mathbb{V}} \sum_{P_k\in\mathbb{P}_{ij}} pSim(P_k, P_k^V)$, where $\mathbb{V}$ is a set of possible vectors and $pSim(\cdot,\cdot)$ is a path similarity function that evaluate the similarity between two paths.

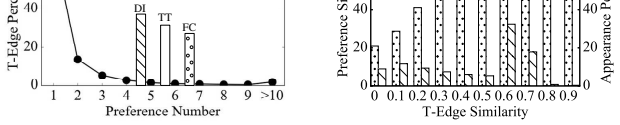
We use a popular path similarity function [22], [32]: $pSim(P_k, P_k^V) = \frac{\sum_{e\in P_k\cap P_k^V} len(e)}{\sum_{e\in P_k} len(e)}$. The intuition is two-fold: first, the more edges the constructed path P_k^V shares with the ground-truth path P_k, the more similar the two paths are; second, the longer the shared edges are, the more similar the two paths are.

A naive way of solving the optimization problem is to search the whole space, i.e., all combinations of features in the master and slave dimensions. However, the search space can be very large, thus rendering the learning algorithm inefficient. We instead propose an efficient learning algorithm that is inspired by coordinate descent. In short, we first identify the best travel cost feature in the master dimension, and next, based on the chosen travel cost feature, we identify the best road condition features in the slave dimension.

Specifically, given the source and destination of each ground truth path P_k, we obtain a lowest-cost path using each cost type. This yields three lowest-cost paths $\hat{P}_k^{DI}$, $\hat{P}_k^{TT}$, and $\hat{P}_k^{FC}$, for distance, travel time, and fuel consumption [33], respectively. We then measure the similarity between path P_k and each of the three lowest-cost paths and choose the optimal cost type whose corresponding lowest-cost path has the highest similarity. Next, we identify the optimal road condition feature. For each road condition feature, we compute a new lowest-cost path based on the optimal cost type while making sure that the road condition feature is also satisfied. We check if the similarity between the new path and the ground truth path P_k can be further improved. The road condition feature that gives the largest improvement is chosen as the optimal road condition feature.

For example, if the optimal cost type is distance, we test if the shortest path with preferences for highways or residential roads can yield a higher similarity compared to shortest path without any road type preferences. If so, we choose the road type that gives the largest similarity improvements. Otherwise, all the road condition features are ignored.

Next, we provide statistical evidence to justify our design choice of choosing only a single representative preference for each T-edge. Given a T-edge (R_i, R_j), we learn a routing preference for each path in $\mathbb{P}_{ij}$, and we count the number of unique preferences. The curve in Figure 5(a) shows that for more than 70% of all T-edges, we obtain a single preference, although multiple paths often exist in $\mathbb{P}_{ij}$. Thus, we chose to learn a single routing preference for each T-edges. On the other hand, Figure 5(a) also suggests that it is possible that a T-edge has more than one preference—we leave the modeling multiple preferences per T-edge as future research.



(a) Distribution of Preferences (b) vs. Preference Similarity

Fig. 5: Statistical Evidences for Design Choices

We also show the distribution of the learned routing preferences as bars in Figure 5(a). We aggregate more than 200 unique routing preferences based on their travel cost features, i.e., DI, TT, and FC. The bars show that the routing preferences are distributed almost uniformly, indicating that T-edges do have different routing preferences.

B. Step 2: Transferring routing preferences

So far, we have identified preference vectors for T-edges. The next step is to associate preference vectors with B-edges, which can then be used to identify appropriate paths for B-edges. To this end, we transfer the routing preferences of T-edges to similar B-edges, which follows the intuition that when two region edges are similar, they also have similar routing preferences. For example, if most local drivers choose the fastest paths with a preference for main roads to travel between a region in the city center and a northern suburb residential area, it is also likely that local drivers have this preference when traveling between another region in the city center and a southern suburb residential area.

Based on the above intuition, we first introduce the similarity function that quantifies the similarity between two region edges and then provide an algorithm that transfers routing preferences between similar region edges.

Similarity between two region edges: Any region edge, a T-edge or a B-edge, connects two regions. A region edge is described by the features of its two regions. In particular, we use two elements dis and $\mathbb{F}$ to describe a region edge re.

Element $re.dis$ is a real value, indicating the Euclidean distance between the centroids of the two regions connected by the region edge. The distance information is an influential factor when drivers choose their paths. For example, drivers may prefer the fastest paths if they travel long distances, but they may prefer the shortest paths when traveling at shorter distances.

Next, element $re.\mathbb{F}$ describes the functionalities of the two regions. Element $re.\mathbb{F}$ is also essential because, for example, when traveling between two business districts and between a residential area and a city center, drivers may have different preferences. In particular, we use a set of road types to describe the functionality of a region [27]. For each region, we consider all edges that are incident to the vertices in the region and select top-k road types of the edges as the region's road type set. For example, regions R_i and R_j have top-2 road type sets {TP1, TP2} and {TP3, TP4}, respectively. Then, region edge (R_i, R_j) has element $re.\mathbb{F}$ that is the Cartesian product of the road type sets from both regions: $re.\mathbb{F} = \{\langle$TP1, TP3$\rangle$, $\langle$TP1, TP4$\rangle$, $\langle$TP2, TP3$\rangle$, $\langle$TP2, TP4$\rangle\}$.

Based on the above, the similarity between two region edges re_i and re_j, quantified by the similarity of their feature vectors, is defined as follows.

$$reSim(re_i, re_j) = \frac{\min(re_i.dis, re_j.dis)}{\max(re_i.dis, re_j.dis)} + J(re_i.\mathbb{F}, re_j.\mathbb{F}).$$

The similarity function is the sum of distance similarity and region function similarity. For distance similarity, the more similar the two distances are, the larger the similarity is. This captures the intuition that travels between equally far apart regions may tend to have similar routing preferences. For region function similarity, we use Jaccard similarity to evaluate the similarity between the region functions. If the two region edges share more function features, meaning that

they connect similar region pairs, travels on the two region edges are expected to have similar routing preferences.

To justify design choices, that (i) similar region edges have similar routing preferences and that (ii) the proposed region edge similarity function $reSim(\cdot,\cdot)$ is effective, we show the results of an experiment using preferences learned from T-edges in Figure 5(b). First, the "Similarity" bars show that similar T-edges have similar routing preferences, while dissimilar T-edges have dissimilar routing preferences. Second, the "Percentage" bars show the percentages of T-edge pairs that fall in a different T-edge similarity ranges. There are many similar (e.g., similarity above 0.5) T-edges, although there are few highly similar (e.g., similarity above 0.9) T-edges. This makes it possible to transfer routing preferences among region edges, and these observations indicate that the design choices are purposeful.

Transferring preferences among similar region edges: We adopt the idea of graph-based transduction learning [34], [35] to transfer routing preferences from T-edges to similar B-edges. First, we build a undirected, weighted graph, where a vertex represents a region edge, which can be a T-edge or a B-edge. Given a total of n region edges, we use an adjacency matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ to record the edge weights of the graph. Specifically, $\mathbf{M}[i,j]$ equals to the similarity $reSim(re_i, re_j)$ between region edges re_i and re_j, where $1 \leqslant i,j \leqslant n$.

Next, we introduce an adjacency matrix reduction threshold amr. In the adjacency matrix, we only keep the values that exceed amr; otherwise, we set the values to 0. This way, the adjacency matrix only captures "sufficiently" similar region edge pairs, which enables to control the accuracy of the transferred preferences. The less dense resulting matrix also improves efficiency (see Figure 8(b) in experiments).

Figure 6 shows a graph with four vertices (i.e., $n = 4$) representing two T-edges and two B-edges. The corresponding matrix $\mathbf{M}$ is also shown. For example, $\mathbf{M}[1,3] = 0.9$ indicates that the similarity between re_1 and re_3 is 0.9, and $\mathbf{M}[2,3] = 0$ indicates that the similarity between re_2 and re_3 is smaller than threshold amr.

In the next step, we use a matrix $\mathbf{Y} \in \mathbb{R}^{n\times p}$ to denote the initial routing preferences of different region edges. Here, n is the total number of region edges, including T-edges and B-edges, and p is the total number of travel cost and road condition features that are used for modeling routing preferences in Section V-A.

To illustrate, we consider two travel cost features DI and TT and three road condition features indicating preferences on road type TP1, TP2, and both, i.e., TP1+2. In this setup, matrix $\mathbf{Y}$ has $p = 5$ columns that represent features DI, TT, TP1, TP2, and TP1+2.

Each row in $\mathbf{Y}$ corresponds to a region edge's routing preference. For a T-edge, the features corresponding to its learned routing preference V^* are set to 1. For example, assuming that T-edge re_1 has preference vector $V^*_{re_1} = \langle$DI, TP1$\rangle$, the first row of $\mathbf{Y}$ is set to $(1,0,1,0,0)$. Similarly, if T-edge re_2 has preference vector $V^*_{re_2} = \langle$TT, TP2$\rangle$, the second row is set to $(0,1,0,1,0)$, as shown in Figure 6. Next, since the routing preferences of the B-edges are unknown, the rows that represent B-edges are set to $(0,0,0,0,0)$.

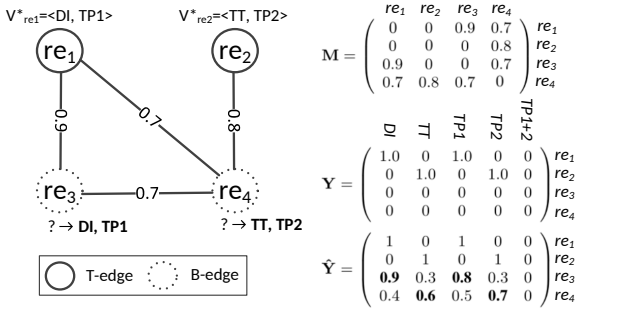


Fig. 6: Transferring routing preferences

The transduction learning yields matrix $\hat{\mathbf{Y}} \in \mathbb{R}^{n\times p}$ that records the transferred routing preferences for the B-edges. Specifically, $\hat{\mathbf{Y}}[i,j]$ indicates the probability of region edge re_i having the j-th routing preference feature. For B-edge re_i, the travel cost feature with the largest probability, i.e., $\arg\max_{x\in\{1,2\}} \hat{\mathbf{Y}}[i,x]$, is used as the final travel cost feature. In the example $\hat{\mathbf{Y}}$ in Figure 6, this is DI for re_3 and TT for re_4. The road type feature with the largest probability, i.e., $\arg\max_{x\in\{3,4,5\}} \hat{\mathbf{Y}}[i,x]$, is used as the final road type feature. In the example $\hat{\mathbf{Y}}$ in Figure 6, this is TP1 for re_3 and TP2 for re_4. Finally, B-edges re_3 and re_4 obtain the transferred routing preferences $V_{re_3} = \langle$DI, TP1$\rangle$ and $V_{re_4} = \langle$TT, TP2$\rangle$, respectively.

Now the remaining question is how to obtain $\hat{\mathbf{Y}}$, which is the core of the transduction learning.

Obtain matrix $\hat{\mathbf{Y}}$: We obtain $\hat{\mathbf{Y}}$ by minimizing the following objective function

$$O(\hat{\mathbf{Y}}) = \sum_{x=1}^{p} \Big[\underbrace{(\mathbf{Y}_{\cdot x} - \hat{\mathbf{Y}}_{\cdot x})^{\mathrm{T}} \mathbf{S} (\mathbf{Y}_{\cdot x} - \hat{\mathbf{Y}}_{\cdot x})}_{\text{Keeping T-edges' preferences}} + \underbrace{\mu_1 \hat{\mathbf{Y}}^{\mathrm{T}}_{\cdot x} \mathbf{L} \hat{\mathbf{Y}}_{\cdot x}}_{\text{Transferring preferences to B-edges}} + \underbrace{\mu_2 ||\hat{\mathbf{Y}}_{\cdot x}||_2^2}_{\text{Regularization}} \Big], \tag{1}$$

where $\mathbf{Y}_{\cdot x}$ and $\hat{\mathbf{Y}}_{\cdot x}$ indicate the x-th column of matrices $\mathbf{Y}$ and $\hat{\mathbf{Y}}$, respectively. Hyper-parameters μ_1 and μ_2 control the relative influences of the second and third terms in the objective function, respectively.

The intuition of each term of the objective function is as follows. First, the T-edges should keep the routing preferences that are learned in step 1. The T-edges' learned routing preferences serve as training data in the transduction learning process.

Matrix $\mathbf{S} \in \mathbb{R}^{n\times n}$ is an auxiliary matrix that indicates which region edges are T-edges. In particular, we organize the region edges such that the first x edges are T-edges and the remaining $n-x$ edges are B-edges. Then, $\mathbf{S}$ is a diagonal matrix, where the first x diagonal entries are set to 1 and the remaining diagonal entries are set to 0. Specifically, we have $\mathbf{S} = \begin{pmatrix} 1&0&0&0\\ 0&1&0&0\\ 0&0&0&0\\ 0&0&0&0 \end{pmatrix}$ in our example because re_1 and re_2 are T-edges.

Based on $\mathbf{S}$, the first term actually computes the sum of the squared differences between $\mathbf{Y}_{\cdot x}$ and $\hat{\mathbf{Y}}_{\cdot x}$ of the rows that

represents T-edges. By minimizing the first term, we try to identify a $\hat{\mathbf{Y}}$ that minimizes the difference. This means that the T-edges should try to keep their learned preferences from step 1. On the other hand, the first term does not pose any constraints between $\mathbf{Y}_{\cdot x}$ and $\hat{\mathbf{Y}}_{\cdot x}$ of the rows that represents B-edges.

Second, the T-edges' routing preferences are transferred to B-edges. The transfer process ensures that the more similar the two region edges are, the more similar their routing preferences are. This is realized by the use of the unnormalized graph Laplacian matrix $\mathbf{L}$ in the second term of Equation 1. In particular, $\mathbf{L} = \mathbf{D} - \mathbf{M}$, where $\mathbf{M}$ is the adjacency matrix and $\mathbf{D}$ is a diagonal matrix where $\mathbf{D}[i,i] = \sum_{k \in \{1 \ldots n\}} \mathbf{M}[i,k]$ and $\mathbf{D}[i,j] = 0$ if $i \neq j$. In our example, we have

$$\mathbf{D} = \begin{pmatrix} 1.6 & 0 & 0 & 0 \\ 0 & 0.8 & 0 & 0 \\ 0 & 0 & 1.6 & 0 \\ 0 & 0 & 0 & 2.2 \end{pmatrix} \quad \mathbf{L} = \begin{pmatrix} 1.6 & 0 & -0.9 & -0.7 \\ 0 & 0.8 & 0 & -0.8 \\ -0.9 & 0 & 1.6 & -0.7 \\ -0.7 & -0.8 & -0.7 & 2.2 \end{pmatrix}$$

With the help of $\mathbf{L}$, the second term actually computes the sum of the products of the similarities of two region edges and the differences of the two region edges' corresponding routing preferences. When the similarity of the two region edges is high, a small difference between their routing preferences make the product significant. Minimizing the second term in the objective function has the effect of smoothly spreading the routing preferences from T-edges to B-edges such that (1) two region edges with high similarities have highly similar routing preferences; (2) two region edges with low similarities may have dissimilar routing preferences.

Third, we conduct L2 regularization [34], [35] to avoid overfitting.

Next, we need to minimize the objective function. By differentiating Equation 1 by $\hat{\mathbf{Y}}_{\cdot x}$ and then setting it to 0, we get

$$(\mathbf{S} + \mu_1 \mathbf{L} + \mu_2 \mathbf{I}) \hat{\mathbf{Y}}_{\cdot x} = \mathbf{S}\mathbf{Y}_{\cdot x} \tag{2}$$

Using basic linear algebra practice, Equation 2 can be solved by iterative approximation algorithms [36], e.g., the Jacobi method [34] or the conjugate gradient method [37]. We need to solve Equation 2 p times; and each time, we obtain a $\hat{\mathbf{Y}}_{\cdot x}$ where $x \in \{1, 2, \ldots, p\}$. Finally, we obtain $\hat{\mathbf{Y}}$.

C. Step 3: Applying Transferred Preferences

After step 2, each B-edge has a transferred preference vector. For each B-edge, we now identify a few appropriate paths according to its transferred preference vector. Consider B-edge (R_i, R_j). Recall that a region has transfer centers where trajectories enter and leave the region (see Section IV-B). For each pair of a transfer center from R_i and a transfer center from R_j, we identify a path according to the preference vector. Finally, the identified paths are associated with B-edge (R_i, R_j).

We proceed to modify Dijkstra's algorithm to accommodate the preference, as shown in Algorithm 1. To ease the discussion, we assume that a B-edge is associated with a transferred routing preference vector ⟨DI, TP1⟩, meaning that minimizing travel distance and using road type TP1 are preferred. Recall that the first dimension is master dimension and the second dimension is the slave dimension.

The overall procedure is similar to the classical Dijkstra's algorithm. In the algorithm, each vertex is associated with two attributes—a cost attribute that records the cost of travel from the source to the vertex and a parent attribute that records the parent vertex of the vertex. And we use a priority queue Q to control the order of visiting different vertices (lines 1–4).

Here, the cost value maintained in a vertex corresponds to the specific cost type feature for the master dimension of a given preference vector. For example, when considering preference vector ⟨DI, TP1⟩, each vertex is associated with a cost that equals to the distance (according to DI) from the source vertex to the vertex.

The algorithm always chooses the vertex with the lowest cost, say vertex u, to continue exploring (line 6). When exploring from u, we differentiate two cases (lines 7–14): (i) at least one edge (u, x) satisfies the slave preference, and (ii) no edge (u, x) exists that satisfies the slave preference. For case (i), only edges that satisfy slave preference are explored. For case (ii), all u's adjacent vertices are explored. This way, we make sure that the preferences on both the master and slave dimensions are accommodated by the algorithm.

Algorithm 1: ApplyingPreferencesModifiedDijkstra

Input: Preference Vector: $V = \langle master, slave \rangle$; Source and destination vertices: v_s, v_d; Road Network: $\mathcal{G}$;
Output: Path P that connects v_s and v_d

1 **for** *each vertex* $v \in \mathcal{G}.\mathbb{V}$ **do**
2 $v.cost \leftarrow +\infty$; $v.parent \leftarrow null$;
3 $v_s.cost \leftarrow 0$;
4 Initialize a priority queue Q and add all vertices to Q;
5 **while** v_d *is still in* Q **do**
6 vertex $u \leftarrow Q.extractMin()$;
7 Boolean $noneSat \leftarrow false$;
8 **if** *there does not exist a vertex* x *such that* x *is* u*'s adjacent vertex and edge* (u, x)*'s road type satisfies* $V.slave$ **then**
9 $noneSat \leftarrow true$;
10 **for** *each vertex* x *that is adjacent* u **do**
11 **if** *edge* (u, x)*'s road type satisfies* $V.slave \vee noneSat$ **then**
12 **if** $u.cost + w_{V.master}(u, x) < x.cost$ **then**
13 $x.cost \leftarrow u.cost + w_{V.master}(u, x)$;
14 $x.parent \leftarrow u$;
15 Construct P from v_d using the parent attributes and return P;

The three steps yield a region graph where each region edge has a set of paths, meaning that the region graph can serve as a foundation for routing.

VI. Routing on Region Graphs

Given an arbitrary pair of a source v_s and a destination v_d in the original road network graph $\mathcal{G}$, we present a routing algorithm that is able to recommend a path connecting them, using the region graph. We distinguish two cases.

Case 1: Vertex v_s is in a region, say R_s, and vertex v_d is also in a region, say R_d. If both vertices are in the same region, i.e., $R_s = R_d$, since we maintain inner-region paths inside regions, we check if trajectories exist that traverse from

v_s to v_d. If yes, we return a path with the largest number of trajectory traversals; if no, we return the fastest path.

If the vertices are not in the same region, i.e., $R_s \neq R_d$, we first identify a region path based on the region graph and then map the region path to a path in the original road network.

Routing on the region graph: The intuition is to find a region path that follows fewer region edges to reach the destination region R_d. This is because if a region path consists of many region edges, it involves the stitching of many paths from different trajectories, which may not represent coherent routing preferences. Thus, in the routing procedure, we always prefer to follow a region edge that enables us to go to a region that is geometrically close to the destination region. When a region edge exists that directly connects R_s and R_d, we always use that region edge. Otherwise, we give higher priorities to the region edges that lead to regions that are closer to the destination region R_d.

To illustrate, consider the region graph shown in Figure 4(b) and assume the physical locations of the regions are also represented in Figure 4(b). Assume that regions R_1 and R_4 are given as the source and destination regions. When exploring from R_1, region R_5 is preferred over regions R_2, R_3 and R_6 because R_5 is much closer to destination region R_4. Finally, the region path $\langle (R_1, R_5), (R_5, R_4) \rangle$ is returned.

Recall that each region edge corresponds to some paths in the original road network graph. Based on this, a region path can be mapped back to a path in the road network graph, which is then returned as the result.

Case 2: At least one of v_s and v_d is not in a region. In this case, we find appropriate region vertices for v_s or/and v_d. Then, we apply the procedure from case 1.

To this end, we issue a fastest path finding algorithm from v_s to v_d based on road network graph $\mathcal{G}$. If a region is visited by the algorithm, we consider it as a candidate region R_s. Similarly, we can identify a candidate region R_d. Then we apply the procedure from case 1 with source region R_s and destination region R_d to identify a path P. Finally, we return a path that consists of three sub-paths—the fastest path from v_s to R_s, denoted as P_s, the path P that connects R_s and R_d, and the fastest path from R_d to v_d, denoted as P_d, as shown in Figure 7. In case there is only one or no candidate region, we simply return the fastest path, e.g., in the case of v_s and v'_d in Figure 7.

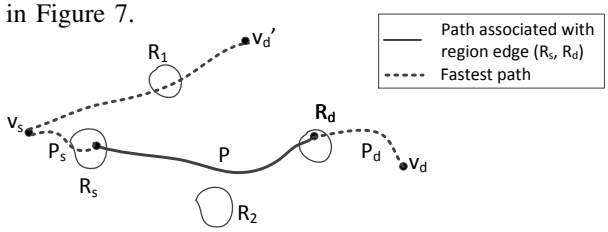


Fig. 7: Routing, Case 2

VII. Empirical Study

We conduct a comprehensive empirical study on two substantial GPS trajectory data sets.

A. Experimental Setup

Road Network and GPS Trajectories: We use two road networks, N_1 and N_2, both obtained from OpenStreetMap (openstreetmap.org). N_1 represents the road network of Denmark and includes 667,950 vertices and 1,636,040 edges, which are contained in a 320 km $\times$ 370 km rectangular region. N_2 represents the road network of Chengdu, China and includes 27,671 vertices and 77,444 edges, which are contained in a 33 km $\times$ 25 km rectangular region.

We use two GPS data sets D_1 and D_2 from N_1 and N_2, respectively. D_1 consists of more than 180 million high-frequency GPS records that were collected by 183 vehicles at 1 Hz (i.e., one GPS record per second) in 2007 and 2008. D_2 consists of 100 million low-frequency GPS records that were collected by 10,864 taxis from August 3rd to 30th 2014. The sampling rate varies from 0.03 Hz to 0.1 Hz. We only use parts of trajectories where taxi have passengers on. We map match [25] the GPS records in D_1 and D_2 onto N_1 and N_2, respectively, obtaining 466,305 trajectories and 185,284 trajectories, where the trajectories in D_2 represent trips with passengers. The travel distance distributions of the trajectories are shown in Table I.

Distance (km)	(0,10]	(10,50]	(50,100]	(100, 500]
# Trajectories of D_1	427,430	35,271	2,263	1,341
Percentage (%)	91.6	7.6	0.5	0.3
Distance (km)	(0,2]	(2,5]	(5,10]	(10, 35]
# Trajectories of D_2	29,256	105,503	43,473	7,052
Percentage (%)	15.8	56.9	23.5	3.8

TABLE I: Statistics of Trajectories

Training and Testing Data: We partition the trajectories in D_1 and D_2 into training data and testing data. Specifically, trajectories that occurred during the first 18 months in D_1 and the first 21 days in D_2 are used as training data; and trajectories that occurred in the last 6 months in D_1 and the last 7 days in D_2 are used as testing data.

Evaluation Criteria: For each trajectory in the testing data *Test*, we record its source and destination, departure time, and the actual path used by the trajectory. Since the aim of the paper is to reuse local drivers' routing intelligence to recommend paths, the paths used by the local drivers are considered as the ground truth (GT) paths.

In the experiments, we run learn-to-route (*L2R*) on each pair of source and destination in *Test*, and we compare the returned path with the GT path, using the path similarity function in Section V-A. In addition, we also identify the shortest path, the fastest path, the paths returned by two personalized routing algorithms and by Google Maps, and compare them with the GT path. The departure time is used when identifying the *L2R* paths, the fastest paths and the Google Maps paths. We also report results w.r.t. the accuracy using a different but also popular path similarity function (see document [31]).

Results are categorized according to the lengths of the GT paths and according to whether the source or destination of a GT path belongs to a region in the obtained region graph. If both the source and destination of a GT path are in regions, the path is in category *InRegion*. If either the source or the

destination is in a region, the path is in category *InOutRegion*. If neither source nor destination belongs to a region, the path is in category *OutRegion*.

Implementation Details: All algorithms are implemented in Java using JDK 1.8. We conduct experiments on a server with a 64-core AMD Opteron(tm) 2.24 GHZ CPU, 528 GB main memory under Ubuntu Linux. We use distance, travel time, and fuel consumption as the travel cost features, where the fuel consumption is computed based on speed limits using vehicular environmental impact models [33]. We use six commonly used road types from OpenStreetMap as road condition features: motorway, trunk, primary, secondary, tertiary, and residential. The transduction learning algorithm for transferring preferences (cf. Section V-B) is implemented using the Junto library (github.com/parthatalukdar/junto).

B. Evaluation of Design Choices

We evaluate the design choices chosen for *L2R*. In particular, we show the effect of important parameters by varying them according to Table II where default values are shown as bold. When we vary one parameter, we keep the remaining parameters at their default values. We show results for both D_1 and D_2 in most empirical studies, but omit some results for D_2 when they show little difference to those of D_1.

Parameters	Values
# T-edges	1X, 2X, 3X, 4X, **5X**
Threshold amr	0.5, 0.6, **0.7**, 0.8, 0.9

TABLE II: Parameters of L2R

Region Sizes: We report the sizes of the obtained regions by computing their convex hulls and then reporting their areas (in km^2) and maximum diameters (in km). Table III reports

Size (km^2)	(0,2]	(2,10]	(10,100]	>100
D_1	3,357 (78.6%) / 9.5	539 (12.6%) / 15.8	304 (7.12%) / 29.9	70 (1.63%) / 304.1
Size (km^2)	(0,2]	(2,5]	(5,10]	>10
D_2	388 (72.1%) / 4.24	127 (23.6%) / 8.17	19 (3.53%) / 8.59	4 (0.74%) / 6.22

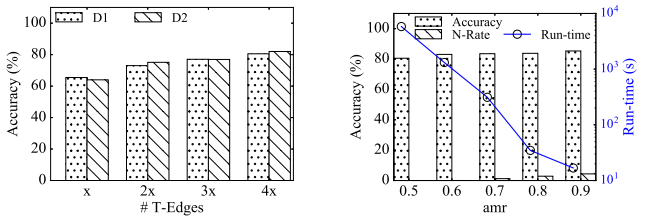
TABLE III: Region Sizes

the numbers of regions whose area falls in given ranges and the maximum diameter of the regions in each range. There are a few large regions, but most regions have sizes less than 2 km^2. This indicates that the proposed modularity-based clustering is able to control the region size and avoids very large regions. D_1 has a few large regions, which represent backbone highways. Since we maintain inner-region paths for regions, large regions do not affect the final routing quality.

Transferring Preferences: We study the accuracy of transferring preferences from T-edges to B-edges in Step 2. As we have no ground-truth preferences for B-edges, we cannot evaluate the accuracy of the transferred preferences in a straightforward manner. To evaluate the accuracy, we randomly partition the preferences of T-edges into 5 partitions. We reserve one partition as a ground truth. Next, we use partition 1; partitions 1 and 2; partitions 1, 2, and 3; and partitions 1, 2, 3, and 4, to conduct the preference transfer. For each T-edge in the reserved partition, we obtain a transferred preference, which we compare it with the ground truth preference. We report the accuracy of the transferred preference against the ground truth preference using Jaccard similarity.

Figure 8(a) shows the accuracy when using 1, 2, 3, and 4 partitions, labeled as X, 2X, 3X, and 4X. The results indicate that the more preferences of T-edges are used, the better the accuracy we get. Therefore, we use all the preferences of T-edges (i.e., 5X) in the remaining experiments.

Next, we consider the effect of the adjacency matrix reduction parameter threshold amr on the transfer process. Since Figure 5(b) already suggests that when the similarity between two region edges is low, their preference vectors are dissimilar, we vary amr from 0.5 to 0.9 and ignore small values. We use 4



(a) Varying # T-Edges (b) Varying amr, D_1

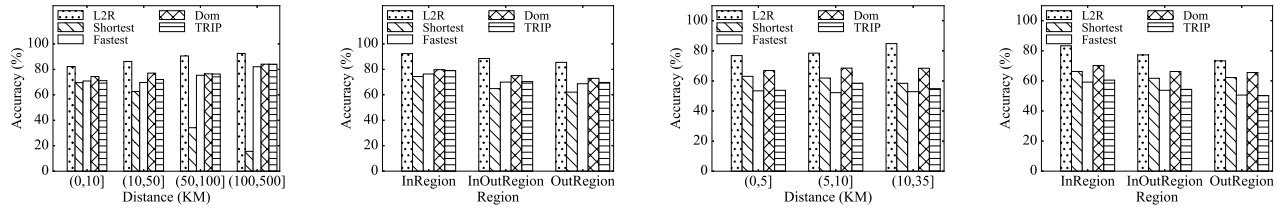
Fig. 8: Parameters of Preference Transfer

partitions of T-edge preferences to build the adjacency matrix and use the last partition as the ground truth preferences. We report the accuracy of the transferred preferences against the ground truth measured using Jaccard similarity, the null rate (N-rate), i.e., the percentage of transferred preferences that get null values, and the run-time in Figure 8(b).

The accuracy of the transfer process increases slightly as amr increases and is not sensitive to the change of amr values when amr exceeds 0.5. This is intuitive because a large amr enables transfer of routing preferences from T-edges only to highly similar B-edges. However, as the amr value increases, the graph used in the graph-based transduction learning may become disconnected. Thus, some B-edges cannot be associated with transferred preferences and thus get a null preference vector. A smaller amr has the effect that the graph used in the graph-based transduction obtains many edges and thus takes longer run-time. The setting $amr = 0.7$ gives the best trade-off, i.e., relatively high accuracy and efficiency and low null rate. We thus use this value in the remaining experiments. We simply associate fastest paths with B-edges with null preference vectors.

C. Comparisons with Other Routing Algorithms

We proceed to compare *L2R* with the shortest and fastest routing algorithms and with two personalized routing algorithms. We apply Dijkstra's algorithm to identify the shortest (*Shortest*) and fastest paths (*Fastest*). We do not apply advanced speeding up techniques for routing, since they have no improvement over the accuracy but only over the query efficiency. When applying such speed-up techniques, the efficiency of computing all paths, including the *L2R* paths, can be improved consistently. We leave such performance improvements as an interesting future research direction.



(a) By Distance, D_1 (b) By Region, D_1 (c) By Distance, D_2 (d) By Region, D_2

Fig. 9: Accuracy

We also consider two personalized routing algorithms, *Dom* [22] and *TRIP* [23], that are able to find personalized "shortest" paths between arbitrary source and destination for individual drivers. The algorithms first learn a global routing preference (rather than a routing preference for each region pair in this paper) for each driver from the driver's historical trajectories, then use the learned preference to obtain new, personalized weights for all edges, and finally apply shortest-path finding using the new edge weights. Specifically, *Dom* utilizes a routing preference that considers distance, travel time, and fuel consumption, whereas *TRIP* uses a routing preference that considers only travel time. In the experiment, we apply each algorithm to learn a routing preference according to a driver's trajectories in the training data. For each trajectory in the testing data, we obtain the source, the destination, and the driver id. Then we apply *Dom* and *TRIP* to compute the personalized, shortest path connecting the source and the destination according to the driver id. Other routing algorithms that use historical trajectories, e.g., [15], [16], [18], do not support routing between arbitrary source and destination, and thus are not comparable to *L2R*.

Accuracy: The accuracies of *L2R*, *Shortest*, *Fastest*, *Dom*, and *TRIP* are calculated using the path similarity function in Section V-A and are reported in Figure 9.

Shortest's accuracy drops as the travel distance increases. This is because *Shortest* tends to find a path that approximates the straight line segment from a source to a destination. Such paths are often not preferred by drivers. In D_1, when traveling longer distances, highways are usually preferred. However, given the fact that using highways often yield longer travel distances, *Shortest* does not return such paths. Therefore, the accuracy of *Shortest* is poor for longer distances.

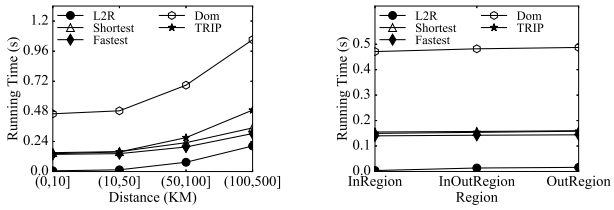
The accuracy of *Fastest* is comparable to that of *Shortest* for small travel distances. However, *Fastest* achieves much higher accuracy when travel distance is longer. When travelling longer distances, highways usually offer the lowest travel times and are therefore returned by *Fastest*. Thus, *Fastest* achieves much better accuracy than does *Shortest*.

Dom achieves higher accuracy than the other routing methods, except *L2R*, because it learns routing preferences that consider the trade-off among distance, travel time, and fuel consumption for individual drivers. However, as it conducts an expensive multi-objective skyline routing process, it requires significantly more running time than other methods (see Figure 10). *TRIP* is slightly more accurate than *Fastest* due to the personal ratio learned for each driver, and it needs similar running time to *Shortest* and *Fastest*.

L2R achieves the highest accuracy in all settings. The accuracy increases as the travel distance becomes longer—this is achieved by capturing the preference for different travel costs and road types in the region graph.

The accuracy of *L2R* decreases when sources and (or) destinations are not in regions. This is intuitive because when no historical trajectories are available for path finding, an *L2R* path simply coincides with the fastest path. However, when historical trajectories can be utilized, *L2R* improves the accuracy of the fastest path (see InOutRegion and OutRegion in Figure 9(b)).

Online Running Time: Run-times are reported in Figure 10. (See document [31] for results of D2). In all settings, *L2R* is most efficient. This is because the path finding process is conducted on the region graph, which is much smaller than the original road network graph. When sources and (or) destinations are not in regions, the run-time of *L2R* increases because it needs extra time to identify the fastest paths from the source (destination) to a region.



(a) By Distance, D_1 (b) By Regions, D_1

Fig. 10: Efficiency

The personalized routing *Dom* requires significantly more running time as it conducts an expensive multi-objective skyline routing process. Next, *Trip* has a running time similar to those of *Shortest* and *Fastest* as all three perform single-objective routing. *Trip* just uses personalized weights.

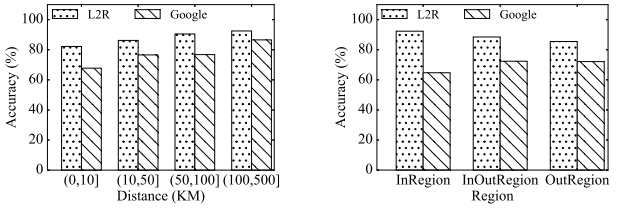
Offline Processing Time for *L2R*: When using all training data and default parameters, the offline processing time for constructing the region graph (Section IV) and for executing steps 1–3 to learn and transfer routing preferences (Section V) for D_1 are 21, 245, 106, and 7 minutes, respectively, and for D_2 are 9, 10, 29, and 0.06 minutes, respectively. Note that such offline processing is parallelizable, e.g., by MapReduce [38].

D. Comparison with Google Maps

We also compare *L2R* with Google Maps. We query the Google Directions API using a source, a destination, and the

departure time from the testing set as arguments to obtain a *Google path*, which consists of a sequence of *waypoints*, represented by longitude-latitude coordinates. We follow an existing methodology [16] to compute the similarity between a Google path and a GT path (details shown in document [31]).

We report the accuracy of Google vs. *L2R* paths in Figure 11 (see document [31] for results on D2). The accuracy of



(a) By distance, D_1 (b) By regions, D_1

Fig. 11: Comparison with Google Maps

Google paths lies between 60% and 85%, and the accuracy increases with the travel distance. However, Google paths show no pattern when we categorize according to whether the source and destination belong to regions. In all settings, *L2R* achieves higher accuracy, indicating that *L2R* has the potential to improve the quality of state-of-the-art routing services.

VIII. Conclusion and Outlook

We propose a learn-to-route solution that enables comprehensive trajectory-based routing. The solution encompasses an algorithm that clusters road intersections into regions, yielding a derived region graph. It learns routing preferences for region pairs with sufficient trajectories and transfers these preferences to region pairs with insufficiently many trajectories. It then utilizes the learned and transferred preferences to enable routing. Empirical studies offer evidence that the solution is practical and is able to compute high-quality routes. In future work, it is of interest to consider finer granularity modeling of time-dependency, e.g., using a time-varying region graph, real-time region graph updates when receiving new trajectories, and the modeling of more than one preference for each T-edge.

References

[1] C. Guo, C. S. Jensen, and B. Yang, "Towards total traffic awareness," *SIGMOD Record*, vol. 43, no. 3, pp. 18–23, 2014.

[2] J. Hu, B. Yang, C. Guo, and C. S. Jensen, "Risk-aware path selection with time-varying, uncertain travel costsa time series approach," *VLDB Journal*, online first, 2018.

[3] Z. Ding, B. Yang, Y. Chi, and L. Guo, "Enabling smart transportation systems: A parallel spatio-temporal database approach," *IEEE Trans. Computers*, vol. 65, no. 5, pp. 1377–1391, 2016.

[4] C. Guo, B. Yang, O. Andersen, C. S. Jensen, and K. Torp, "Ecosky: Reducing vehicular environmental impact through eco-routing," in *ICDE*, 2015, pp. 1412–1415.

[5] M. Hua and J. Pei, "Probabilistic path queries in road networks: traffic uncertainty aware path selection," in *EDBT*, 2010, pp. 347–358.

[6] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k shortest paths with diversity," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 3, pp. 488–502, 2018.

[7] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang, "Stochastic skyline route planning under time-varying uncertainty," in *ICDE*, 2014, pp. 136–147.

[8] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k optimal sequenced routes," in *ICDE*, 2018, p. 12 pages.

[9] J. Dai, B. Yang, C. Guo, C. S. Jensen, and J. Hu, "Path cost distribution estimation using trajectory data," *PVLDB*, vol. 10, no. 3, pp. 85–96, 2016.

[10] B. Yang, J. Dai, C. Guo, and C. S. Jensen, "PACE: A PAth-CEntric paradigm for stochastic path finding," *VLDB Journal*, online first, 2017.

[11] S. Aljubayrin, B. Yang, C. S. Jensen, and R. Zhang, "Finding non-dominated paths in uncertain road networks," in *SIGSPATIAL*, 2016, pp. 15:1–15:10.

[12] J. Hu, B. Yang, C. S. Jensen, and Y. Ma, "Enabling time-dependent uncertain eco-weights for road networks," *GeoInformatica*, vol. 21, no. 1, pp. 57–88, 2017.

[13] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *WEA*, 2008, pp. 319–333.

[14] V. Ceikute and C. S. Jensen, "Routing service quality - local driver behavior versus routing services," in *MDM*, 2013, pp. 97–106.

[15] Z. Chen, H. T. Shen, and X. Zhou, "Discovering popular routes from trajectories," in *ICDE*, 2011, pp. 900–911.

[16] V. Ceikute and C. S. Jensen, "Vehicle routing with user-generated trajectory data," in *MDM*, 2015, pp. 14–23.

[17] W. Luo, H. Tan, L. Chen, and L. M. Ni, "Finding time period-based most frequent path in big trajectory data," in *SIGMOD*, 2013, pp. 713–724.

[18] J. Dai, B. Yang, C. Guo, and Z. Ding, "Personalized route recommendation using big trajectory data," in *ICDE*, 2015, pp. 543–554.

[19] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. P. Sondag, "Adaptive fastest path computation on a road network: A traffic mining approach," in *VLDB*, 2007, pp. 794–805.

[20] L. Wei, Y. Zheng, and W. Peng, "Constructing popular routes from uncertain trajectories," in *SIGKDD*, 2012, pp. 195–203.

[21] A. Balteanu, G. Jossé, and M. Schubert, "Mining driving preferences in multi-cost networks," in *SSTD*, 2013, pp. 74–91.

[22] B. Yang, C. Guo, Y. Ma, and C. S. Jensen, "Toward personalized, context-aware routing," *VLDB Journal*, vol. 24, no. 2, pp. 297–318, 2015.

[23] J. Letchner, J. Krumm, and E. Horvitz, "Trip router with individualized preferences (TRIP): incorporating personalization into route planning," in *AAAI*, 2006, pp. 1795–1800.

[24] D. Delling, A. V. Goldberg, M. Goldszmidt, J. Krumm, K. Talwar, and R. F. Werneck, "Navigation made personal: inferring driving preferences from GPS traces," in *SIGSPATIAL*, 2015, pp. 31:1–31:9.

[25] P. Newson and J. Krumm, "Hidden Markov map matching through noise and sparseness," in *SIGSPATIAL*, 2009, pp. 336–343.

[26] X. Liang, J. Zhao, L. Dong, and K. Xu, "Unraveling the origin of exponential law in intra-urban human mobility," *arXiv:1305.6364*, 2013.

[27] G. Forbes, "Urban roadway classification," in *Urban Street Symposium*, 1999.

[28] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, p. 026113, 2004.

[29] M. E. Newman, "Analysis of weighted networks," *Physical review E*, vol. 70, no. 5, p. 056131, 2004.

[30] H. Shiokawa, Y. Fujiwara, and M. Onizuka, "Fast algorithm for modularity-based graph clustering," in *AAAI*, 2013, pp. 1170–1176.

[31] C. Guo, B. Yang, J. Hu, and C. S. Jensen, "Learning to route with sparse trajectory sets—extended version," *CoRR*, vol. abs/1802.07980, 2018.

[32] E. Erkut and V. Verter, "Modeling of transport risk for hazardous materials," *Operations Research*, vol. 46, no. 5, pp. 625–642, 1998.

[33] C. Guo, B. Yang, O. Andersen, C. S. Jensen, and K. Torp, "Ecomark 2.0: empowering eco-routing with vehicular environmental models and actual vehicle fuel consumption data," *GeoInformatica*, vol. 19, no. 3, pp. 567–599, 2015.

[34] P. P. Talukdar and K. Crammer, "New regularized algorithms for transductive learning," in *ECML/PKDD*, 2009, pp. 442–457.

[35] Y. Wang, B. Yang, L. Qu, M. Spaniol, and G. Weikum, "Harvesting facts from textual web sources by constrained label propagation," in *CIKM*, 2011, pp. 837–846.

[36] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[37] B. Yang, M. Kaul, and C. S. Jensen, "Using incomplete information for complete weight annotation of road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 5, pp. 1267–1279, 2014.

[38] B. Yang, Q. Ma, W. Qian, and A. Zhou, "TRUSTER: trajectory data processing on clusters," in *DASFAA*, 2009, pp. 768–771.