2020 IEEE 36th International Conference on Data Engineering (ICDE)

Learning to Rank Paths in Spatial Networks

Sean Bin Yang and Bin Yang
Department of Computer Science, Aalborg University, Denmark
{seany, byang}@cs.aau.dk

***Abstract*—Modern navigation services often provide multiple paths connecting the same source and destination for users to select. Hence, ranking such paths becomes increasingly important, which directly affects service quality. We present *PathRank*, a data-driven framework for ranking paths based on historical trajectories. If a trajectory used path P from source s to destination d, *PathRank* considers this as an evidence that P is preferred over all other paths from s to d. Thus, a path that is similar to P should have a larger ranking score than a path that is dissimilar to P. Based on this intuition, *PathRank* models path ranking as a regression problem that assigns each path a ranking score. We first propose an effective method to generate a compact set of diversified paths using trajectories as training data. Next, we propose an end-to-end deep learning framework to solve the regression problem. In particular, a spatial network embedding is proposed to embed each vertex to a feature vector by considering the road network topology. Since a path is represented by a sequence of vertices, which is now a sequence of feature vectors after embedding, recurrent neural network is applied to model the sequence. Empirical studies on a substantial trajectory data set offer insight into the designed properties of the proposed framework and indicating that it is effective and practical.**

I. Introduction

Vehicular transportation reflects the pulse of a city. It not only affects people's daily lives and also plays an essential role in many businesses as well as society as a whole [1]. A fundamental functionality in vehicular transportation is routing [2]. Given a source and a destination, classic routing algorithms, e.g., Dijkstra's algorithm, identify an optimal path connecting the source and the destination, where the optimal path is often the path with the least travel cost, e.g., the shortest path or the fastest path. However, a routing service quality study [3] shows that local drivers often choose paths that are neither shortest nor fastest, rendering classic routing algorithms often impractical in many real world routing scenarios.

To contend with this challenge, a wide variety of advanced routing algorithms, e.g., skyline routing [4] and k-shortest path routing [5], [6], are proposed to identify a set of optimal paths, where the optimality is defined based on, e.g., pareto optimality or top-k least costs, which provide drivers with multiple candidate paths. In addition, commercial navigation systems, such as Google Maps and TomTom, often follow a similar strategy by suggesting multiple candidate paths to drivers, although the criteria for selecting the candidate paths are often confidential.

Under this context, ranking the candidate paths is essential for ensuring high routing quality. Existing solutions often rely on simple heuristics, e.g., ranking paths w.r.t. their travel times. However, travel times may not always be the most important factor when drivers choose paths, as demonstrated in the routing quality study where drivers often do not choose the fastest paths [3]. In addition, existing solutions often provide the same ranking to all users but ignore distinct preferences which different drivers may have.

In this paper, we propose a data-driven ranking framework *PathRank*, which ranks candidate paths by taking into account the paths used by local drivers in their historical trajectories. More specifically, *PathRank* models ranking candidate paths as a "regression" problem—for each candidate path, *PathRank* estimates a ranking score for the candidate path.

We first prepare a training data set using historical trajectories. For each trajectory, we consider the path used by the trajectory, i.e., trajectory path, as the ground truth path and thus has the highest ranking score, say 1. We then propose an effective method to generate a diversified training path set for the trajectory path, where each training path is associated with a ranking score that equals to the similarity between the training path and the trajectory path. The intuition behind is that if a training path is similar to the ground truth trajectory path, it should also have a higher ranking score.

Next, we propose a deep learning framework to learn meaningful feature representations of paths, which enables effective ranking. The input for *PathRank* is a path and its label is a similarity score. In order to use deep learning to solve the similarity score regression problem, a prerequisite is to represent the input path into an appropriate feature space. To this end, we propose to use a vertex embedding network to transfer each vertex in the input path to a feature vector. Since a path is a sequence of vertices, after vertex embedding, the path becomes a sequence of feature vectors. Then, since RNNs are capable of capturing dependency for sequential data, we employ an RNN to model the sequence of feature vectors. The RNN finally outputs an estimated similarity score, which is compared against the ground truth similarity.

To the best of our knowledge, this is the first data-driven, end-to-end solution for ranking paths in spatial networks.

II. Related Work

A. Learning to rank

Learning to rank plays an important role in ranking in the context of information retrieval (IR), where the primary goal is to learn how to rank documents or web pages w.r.t. queries, which are all represented as feature vectors. Learning

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00225

to rank methods in IR can be categorized into point-wise [7], pair-wise [8], and list-wise [9] methods. In particular, point-wise methods estimate a ranking score for each individual document. Then, the documents can be ranked based on the ranking scores [7]. We follow the idea of the point-wise learning to rank techniques and propose *PathRank* to rank paths in spatial networks while considering road network topology.

B. Network Representation Learning

Network representation learning, a.k.a., graph embedding, aims to learn low-dimensional feature vectors for vertices while preserving network topology structure such that the vertices with similar feature vectors share similar structural properties [10], [11]. A representative method is DeepWalk [11]. DeepWalk first samples sequences of vertices based on truncated random walks, where the sampled vertex sequences capture the connections between vertices in the graph. Then, skip-gram model [12] is used to learn low-dimensional feature vectors based on the sampled vertex sequences. Node2vec [10] considers higher order proximity between vertices by maximizing the probability of occurrences of subsequent vertices in fixed length random walks. A key difference from DeepWalk is that node2vec employs biased-random walks that provide a trade-off between breadth-first and depth-first searches, and hence achieves higher quality and more informative embedding than DeepWalk does. We use node2vec in our experiments.

C. Machine Learning for Route Recommendation

Machine learning has been applied to improve route recommendation [1], [13]. Personalized routing aims to identify the best path for a specific driver but cannot rank a set of candidate paths [14]. Multitask learning is applied to model different drivers' driving behavior [15], but cannot be used directly for ranking paths. Additional attempts have been made for estimating travel time or fuel consumption distributions [16]–[19], which are also different from ranking.

III. Preliminaries

A. Basic Concepts

A *road network* is modeled as a weighted, directed graph $G = (\mathbb{V}, \mathbb{E}, D, T, F)$. Vertex set $\mathbb{V}$ represents road intersections; edge set $\mathbb{E} \subset \mathbb{V} \times \mathbb{V}$ represents road segments. Functions D, T, and F maintain distances, travel times, and fuel consumption of traversing the edges in graph G.

A *path* $\boldsymbol{P} = (v_1, v_2, v_3, \ldots, v_X)$ is a sequence of vertices and each two adjacent vertices must be connected by an edge in $\mathbb{E}$. A *trajectory* $\boldsymbol{T} = (p_1, p_2, p_3, \ldots, p_Y)$ is a sequence of GPS records pertaining to a trip, where each GPS record $p_i = (location, time)$ represents the location of a vehicle at a particular timestamp. Map matching is able to map a GPS record to a specific location on an edge in the underlying road network, thus aligning a trajectory with a path in the underlying road network. We call such paths *trajectory paths*.

We use the weighted Jaccard Similarity [1] to evaluate the similarity between two paths.

IV. Training Data Generation

We proceed to elaborate how to generate a set of training paths for a trajectory path P from source s to destination d.

A naive way to generate the training path set is to simply include all paths from s to d. This is infeasible to use in real world settings since the training path set may contain a huge number of paths in a city-level road network graph, which in turn makes the training prohibitively inefficient. Thus, we aim to identify a *compact* training path set, where only a small number of paths, e.g., less than 10 paths, are included.

To this end, the classic top-k shortest path algorithm, e.g. Yen's algorithm [5], could be employed to choose top-9 shortest paths connecting s and d as the training paths. However, a serious issue of this strategy is that the top-k shortest paths are often highly similar. Thus, their similarities w.r.t. the ground truth, trajectory path P, are also similar, which adversely affect the effectiveness of the subsequent ranking score regression—if the similarities of training paths only spread over a small range, they only provide training instances for estimating ranking scores in the small range, which may make the trained model unable to make accurate estimations for ranking scores outside the small range. Thus, an ideal strategy should be providing a set of training paths whose similarities cover a large range.

To this end, we propose the second strategy using the diversified top-k shortest paths [6]. We first include the shortest path into the result. Next, we check the next shortest path P' and only include P' into the result if P' is dissimilar with all the existing paths in the result. In other words, if the similarity between P' and an existing path in the result is greater than a threshold, P' is not included into the result set. We keep this procedure until we find k paths or we examine all paths from s to d.

For each trajectory path P, we identify a set of diversified shortest paths $\{P'_i\}$. Then, $\{P'_i, sim_i\}$ serves the training data set where the similarity scores $sim_i = WeightedJaccard(P, P'_i)$ are considered as labels.

V. PathRank

We propose an end-to-end deep learning framework *PathRank* to estimate similarity scores for paths. As shown in Figure 1, *PathRank* consists of two neural networks—a vertex embedding network and a recurrent neural network (RNN).

A. Vertex Embedding

We represent a vertex v_i in road network graph G as a one-hot vector $q_i \in \mathbb{R}^N$, where N represents the number of vertices in G, i.e., $N = |G.\mathbb{V}|$. Specifically, the i-th vertex v_i in graph G is represented as a vector q_i where the i-th bit is 1 and the other $N-1$ bits are 0.

Vertex embedding employs an embedding matrix $B \in \mathbb{R}^{M \times N}$ to transfer a vertex's one-hot vector q_i into a new feature vector $x_i = Bq_i \in \mathbb{R}^M$. The feature vector is often in a smaller space, where $M < N$.

Graph embedding aims at learning low dimensional, latent representations of vertices in a graph by taking into account the

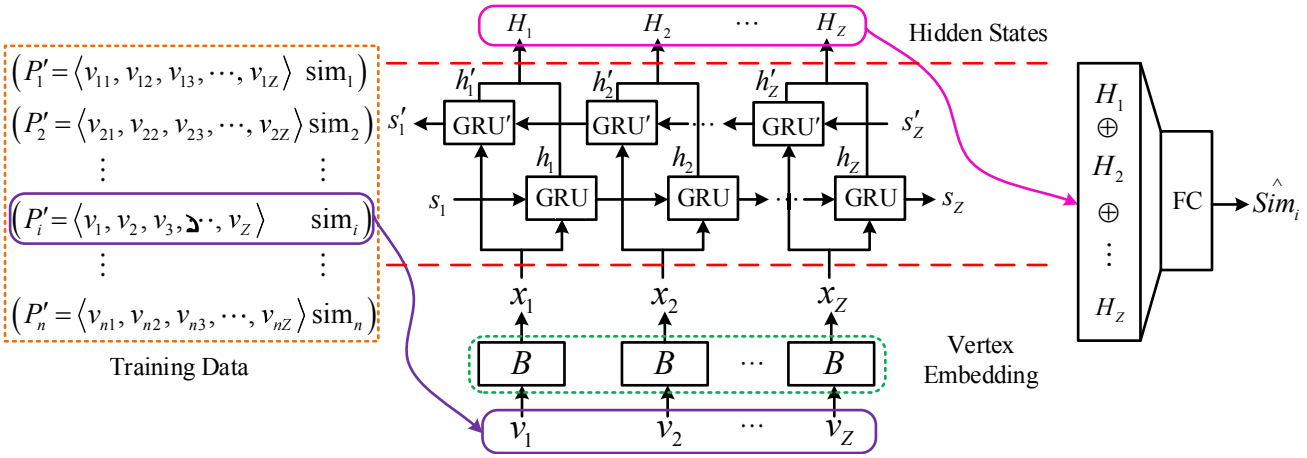

Fig. 1: *PathRank* Overview.

graph topology. Here, we apply an existing graph embedding method, node2vec [10], to initialize embedding matrix B so that the road network topology is well captured. In addition, during training, we allow *PathRank* to update B to better fit the similarity score regression task.

Given a training path $P'_i = \langle v_1, v_2, \ldots, v_Z \rangle$, we apply the same embedding matrix B to transfer each vertex to a feature vector. Thus, the training path P is represented as a sequence of features $\langle x_1, x_2, \ldots, x_Z \rangle$, where $x_j = Bq_j$ and $1 \leq j \leq Z$.

B. RNN

The feature sequence represents the flow of travel on path P'_i and we would like to capture the flow. To this end, we fed the feature sequence $\langle x_1, x_2, \ldots, x_Z \rangle$ into a recurrent neural network, which is known to be effective for modeling sequences. Specifically, we employ a bidirectional gated recurrent neural network (BD-GRU) [20] to capture the sequential dependencies in both the direction and the opposite direction of the travel flow.

We consider the direction of the travel flow first, i.e., from left to right. A GRU unit learns sequential correlations by maintaining a hidden state $\mathbf{h}_j \in \mathbb{R}^M$ at position j, which can be regard as an accumulated information of the positions on the left of position j. Specifically, $h_j = GRU(x_j, \mathbf{h}_{j-1})$, where x_j is the input feature vector at position j and $\mathbf{h}_{j-1}$ is the hidden state at position $j-1$, i.e., the hidden state of the left position.

For the opposite direction of the travel flow, i.e., from right to left, we apply another GRU to generate hidden state $\mathbf{h}'_j = GRU'(x_j, \mathbf{h}'_{j+1})$. Here, the input consists of the feature vector at position j and the hidden state at position $j+1$, i.e., the right hidden state. The final hidden state H_j at position j is the concatenation of the hidden states from both GRUs, i.e., $H_j = \mathbf{h}_j \oplus \mathbf{h}'_j$ where $\oplus$ indicates the concatenation operation.

C. Fully Connected Layer

We stack all outputs from the BD-GRU units into a long feature vector $f_i = \langle H_1 \oplus H_2 \oplus \ldots \oplus H_Z \rangle$ where $\oplus$ indicates the concatenation operation. Then, we apply a fully connected layer with weight vector $W_{FC} \in \mathbb{R}^{|f_i| \times 1}$ to produce a single value $\hat{sim}_i = f_i^T W_{FC}$, as the estimated similarity for the training path P'_i.

D. Loss Function

The loss function of the basic framework is shown in Equation 1.

$$\mathcal{L}(\mathbf{W}) = \frac{1}{|n|} \sum_{i=1}^{n} \left(\hat{sim}_i - sim_i \right)^2 + \lambda \|\mathbf{W}\|_2^2 \qquad (1)$$

The first term of the loss function measures the discrepancy between the estimated similarity $\hat{sim}_i$ and the ground truth similarity sim_i. We use the average of square error to measure the discrepancy, where n is the total number of training paths we used for training.

The second term of the loss function is a L2 regularizer on all learnable parameters in the model, including the embedding matrix B, multiple matrices used in BD-GRU, and the matrix in the final fully connected layer W_{FC}. Here, λ controls the relative importance of the second term w.r.t. the first term.

VI. Experiments

A. Experiments Setup

1) Road Network and Trajectories: We consider the road network in North Jutland, Denmark. We use a substantial GPS data set occurred on the road network, which consists of 180 million GPS records for a two-year period from 183 vehicles. The sampling rate of the GPS data is 1 Hz (i.e., one GPS record per second). We split the GPS records into 22,612 trajectories representing different trips. We map match the GPS trajectories, such that for each trajectory, we obtain its corresponding trajectory path.

2) Ground Truth Data: For each trajectory T, we obtain its source s, destination d, and the trajectory path P_T. Then, we employ two different strategies to generate two sets of training paths according to the source-destination pairs (s, d)—top-k shortest paths (*TkDI*) and diversified top-k shortest paths

(*D-TkDI*). For each training path P, we employ weighted Jaccard similarity $WeightedJaccard(P,\ P_T)$ as P's *ground truth* ranking score.

3) Evaluation Metrics: We evaluate the accuracy of the proposed *PathRank* framework based on two categories of metrics. The first category includes Mean Absolute Error (MAE) and Mean Absolute Relative Error (MARE). The second category includes Kendall rank correlation coefficient (denoted by τ) and Spearman's rank correlation coefficient (denoted by ρ), which measure the similarity, or consistency, between a ranking based on the estimated ranking scores and a ranking based on the ground truth ranking scores.

B. Experimental Results

We investigate how the different training data generation strategies affect the accuracy of *PathRank*. We first consider PR-A1, where we only use graph embedding method node2vec to initialize the vertex embedding matrix B and do not update B during training.

Table I shows the results, where we categorize the training data generation strategies into two categories based on top-k shortest paths (*TkDI*) and diversified top-k shortest paths (*D-TkDI*). For each category, the best results are highlighted with underline. The best results overall is also highlighted with bold. We also show results when the embedding feature sizes are $M = 64$ and $M = 128$, respectively. The results show that (1) when using the diversified top-k paths for training, we have higher accuracy (i.e., lower MAE and MARE and larger τ and ρ) compared to when using top-k paths, suggesting that using diversified paths is essential for accuracy; (2) a larger embedding feature size M achieves better results.

TABLE I: Training Data Generation Strategies, *PR-A1*

Strategies	M	**MAE**	**MARE**	τ	ρ
TkDI	64	0.1433	0.2300	0.6638	0.7044
	128	0.1168	0.1875	0.6913	0.7330
D-TkDI	64	0.1140	0.1830	0.6959	0.7346
	128	**0.0955**	**0.1533**	**0.7077**	**0.7492**

Next, we consider PR-A2, where the graph embedding matrix B is also updated during training to fit better the ranking score regression problem. Table II shows the results. The two observations from Table I also hold for Table II. In addition, PR-A2 achieves better accuracy than does *PR-A1*, meaning that updating embedding matrix B is useful.

TABLE II: Training Data Generation Strategies, *PR-A2*

Strategies	M	**MAE**	**MARE**	τ	ρ
TkDI	64	0.1163	0.1868	0.6835	0.7256
	128	0.1130	0.1814	0.7082	0.7481
D-TkDI	64	0.0940	0.1509	0.7144	0.7532
	128	**0.0855**	**0.1373**	**0.7339**	**0.7731**

VII. Conclusion and Future work

We propose *PathRank*, a learning to rank technique for ranking paths in spatial networks. We propose an effective way to generate a set of training paths to enable effective and efficient learning. Then, we propose and end-to-end deep learning framework to enable graph embedding that takes into account road network topology. A recurrent neural network, together with the learned graph embedding, is employed to estimate the ranking scores which eventually enable ranking paths. Empirical studies conducted on a large real world trajectory set demonstrate that *PathRank* is effective and efficient for practical usage. As future work, it is of interest to apply outlier detection algorithms [21] to filter abnormal trajectories to improve the ranking quality of *PathRank*.

References

[1] C. Guo, B. Yang, J. Hu, and C. Jensen, "Learning to route with sparse trajectory sets," in *ICDE*, 2018, pp. 1073–1084.

[2] S. A. Pedersen, B. Yang, and C. S. Jensen, "Fast stochastic routing under time-varying uncertainty." *VLDB Journal*, to appear, 2019.

[3] V. Ceikute and C. S. Jensen, "Routing service quality–local driver behavior versus routing services," in *MDM*, vol. 1, 2013, pp. 97–106.

[4] 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.

[5] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[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] F. C. Gey, "Inferring probability of relevance using the method of logistic regression," in *SIGIR'94*, 1994, pp. 222–231.

[8] L. Rigutini, T. Papini, M. Maggini, and F. Scarselli, "Sortnet: Learning to rank by a neural-based sorting algorithm," in *SIGIR*, vol. 42, no. 2, 2008, pp. 76–79.

[9] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *ICML*, 2007, pp. 129–136.

[10] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *SIGKDD*, 2016, pp. 855–864.

[11] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *SIGKDD*, 2014, pp. 701–710.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[13] J. Hu, B. Yang, C. Guo, and C. S. Jensen, "Risk-aware path selection with time-varying, uncertain travel costs: a time series approach," *VLDB J.*, vol. 27, no. 2, pp. 179–200, 2018.

[14] 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.

[15] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Distinguishing trajectories from different drivers using incompletely labeled trajectories," in *CIKM*, 2018, pp. 863–872.

[16] 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.

[17] J. Hu, C. Guo, B. Yang, and C. S. Jensen, "Stochastic weight completion for road networks using graph convolutional networks," in *ICDE*, 2019, pp. 1274–1285.

[18] B. Yang, J. Dai, C. Guo, C. S. Jensen, and J. Hu, "PACE: a path-centric paradigm for stochastic path finding," *VLDB J.*, vol. 27, no. 2, pp. 153–178, 2018.

[19] S. A. Pedersen, B. Yang, and C. S. Jensen, "A hybrid learning approach to stochastic routing," in *ICDE*, to appear, 2020.

[20] S. Mangal, P. Joshi, and R. Modak, "Lstm vs. gru vs. bidirectional rnn for script generation," *arXiv preprint arXiv:1908.04332*, 2019.

[21] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Outlier detection for time series with recurrent autoencoder ensembles," in *IJCAI*, 2019, pp. 2725–2732.