A Learning-based Method for Computing Shortest Path Distances on Road Networks

Shuai Huang, Yong Wang, Tianyu Zhao, Guoliang Li
Department of Computer Science, Tsinghua University, Beijing, China
huang-s19,wangy18,zhaoty17@mails.tsinghua.edu.cn,liguoliang@tsinghua.edu.cn

***Abstract*—Computing the shortest path distances between two vertices on road networks is a core operation in many real-world applications, e.g., finding the closest taxi/hotel. However existing techniques have several limitations. First, traditional Dijkstra-based methods have long latency and cannot meet the high-performance requirement. Second, existing indexing-based methods either involve huge index sizes or have poor performance. To address these limitations, in this paper we propose a learning-based method which can efficiently compute an approximate shortest-path distance such that (1) the performance is super fast, e.g., taking 60-150 nanoseconds; (2) the error ratio of the approximate results is super small, e.g., below 0.7%; (3) scales well to large road networks, e.g., millions of nodes. The key idea is to first embed the road networks into a low dimensional space for capturing the distance relations between vertices, get an embedded vector for each vertex, and then perform a distance metric (L_1 metric) on the embedded vectors to approximate shortest-path distances. We propose a hierarchical model to represent the embedding, and design an effective method to train the model. We also design a fine-tuning method to judiciously select high-quality training data. Extensive experiments on real-world datasets show that our embedding based approach significantly outperforms the state-of-the-art methods.**

I. Introduction

Computing the shortest-path distance between two vertices on road networks is a core operation in many location-based applications. For example, Uber finds the closest car for a passenger and requires to compute the shortest-path distances between multiple cars and the passenger. Suppose there are 10K passengers that pose requests in each second and there are 1K candidate cars for each passenger. Then Uber requires to compute the shortest-path distances for 10 million pairs. Even if it takes 1 microsecond for computing the shortest-path distance of a pair, Uber still takes 10 seconds for processing these requests. Thus it is rather crucial to improve the performance of computing the shortest-path distances. Considering another example, Yelp supports to find k-nearest points of interests (POIs) for users, and Yelp also requires to efficiently compute the shortest-path distances between POIs and users.

There are many studies to efficiently compute the shortest-path distance on road networks [2], [5], [11]–[13], [16], [20]–[22], [26], [32]–[34]. Dijkstra [10] is a classic method which judiciously computes the shortest path by pruning unpromising roads, but it incurs unacceptable time decay (in several seconds) on large road networks. Recently there are some index-based methods that utilize the indexes to efficiently compute the shortest-path distances, *e.g.,* CH [11] and H2H [22] are exact methods; ACH [12] and Distance Oracle [27] use lower time cost to compute approximate results. However, these methods either incur huge pre-processing time (*e.g.,* CH, ACH) or cannot scale to large road networks (*e.g.,* Distance Oracle) or take large index sizes (*e.g.,* H2H). Moreover, they are not fast enough to support instant feedback.

To address these limitations, in this paper we propose a learning-based method which can efficiently compute an approximate shortest-path distance such that (1) the performance is super fast, *e.g.,* taking 60-150 nanoseconds, 10x faster than the existing fastest exact algorithm; (2) the error ratio of the approximate results is super small, *e.g.,* below 0.7%, which significantly outperforms the existing approximate algorithms under the same condition; (3) scales well to large road networks, *e.g.,* millions of nodes. This paper is motivated by the fact that the *distance relations*, *i.e.,* shortest-path distances, between vertices in a planar road network can be captured in latent space with a well-suited representation metric (L_1). It is inspired by but different from the existing node embedding methods such as DeepWalk [23] and LINE [29] that capture *social relations*, *e.g.,* neighborhood similarity, between nodes.

We propose a road network embedding (**RNE**) model that embeds 2-dimensional vertices (longitudes and latitudes) on road networks into d-dimensional embedding vectors, with which L_1 metric can be utilized to approximate the shortest-path distances, instead of performing search on the graph (as Figure 1 shows). There are three challenges. The first is to design an effective representation model that can capture the distance relationship between vertices spreading all over a road network. We propose a hierarchical embedding model. The second is to efficiently and effectively train the embedding model. We propose a hierarchical method for embedding training. The third is to select high-quality training samples to achieve low estimation errors. We design a fine-tuning method to judiciously select high-quality training data.

Our main contributions are summarized as follows.

- We propose a learning-based method to efficiently compute the approximate shortest-path distances on road networks. We embed the road network and utilize the L_1 metric to capture the shortest-path distances (see Section III).

*Guoliang Li is the corresponding author. This work was supported by NSF of China (61925205, 61632016), Huawei, and TAL education.

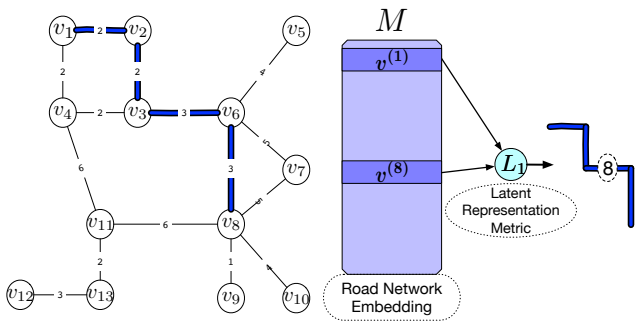


Fig. 1: An example of road network and its network-distance embedding

- We propose a hierarchical learning strategy that not only improves the embedding performance (*i.e.,* improving the accuracy of approximating the shortest-path distances) but also accelerates the training process (see Section IV).
- We propose an active fine-tuning method, which can judiciously select high-quality training data to address the under-fitting problem (see Section V).
- We extend our embedding model to support efficient kNN queries and range queries on road networks (see Section VI).
- We have conducted extensive experiments on real-world datasets and compared with state-of-the-arts. Experimental results showed that our embedding based method significantly outperformed existing approaches and is practical for usage (see Section VII).

II. Related Work

A. Shortest Path on Road Network

There are extensive studies on shortest-path search, which utilize well-designed indexes to accelerate the searching process [1], [4], [31].

Exact Algorithms. Contraction Hierarchies (CH) [11] computes an importance order on vertices and adds shortcuts based on the importance order to support fast shortest paths searches. G-Tree [30], [35], [36] builds a hierarchical balance tree index, which searches for the shortest distance by traversing the corresponding tree nodes. Ouyang *et al.* design H2H [22] which combines tree decomposition hierarchy and hop-based labeling, and achieve high performance.

Approximate Algorithms. ACH [12] is a heuristic optimization over CH. It stops adding shortcuts when a replacement path is a bit longer, leading to an error bounded shortest distance. Distance oracle [27] divides all the vertex pairs into well-separated groups and retrieves the ϵ-approximate shortest distances of any vertex pairs in $O(\log |V|)$ time. ALT [13] generates a set of landmarks U and a $|U| \times |V|$ label matrix, with which it utilizes the triangle inequality to estimate the shortest distances in $O(|U|)$ time (we denote this estimation method as LT), and then performs A^* to search the shortest path.

TABLE I: Table of Notations.

Notation	Definition
$G = (V, E, W)$	a road network graph
$\phi(v_s, v_t), \hat{\phi}(v_s, v_t)$	(approximate) shortest distance from v_s to v_t
$M = \{\boldsymbol{v}^{(i)}\}$	$\lvert V\rvert \times d$ vertices embedding (global) matrix
$\bar{M}_l = \{\bar{\boldsymbol{v}}_l^{(i)}\}$	local embedding matrix in level l ($\{v_{l,i}\}$)
$S = \{(v_s, v_t, \phi_{(v_s,v_t)})\}$	samples of vertex pairs and shortest distances
$P_l = \{G_{li}\}$	set of sub-graphs in level l
U	set of landmarks

kNN Algorithms. SILC [25] pre-computes the shortest paths between all pairs of vertices. V-tree [28] extends G-tree by additionally indexing distances between objects and border vertices.

Different from these traditional methods, RNE is a learning method which trains a hierarchical embedding for vertices and then utilizes the embedding to efficiently compute the approximate shortest distances, instead of performing search. Since the learned embedding captures the shortest distances in latent space, with a metric which is well-suited for distance representation of road networks, on both the efficiency and the accuracy, it significantly outperform Distance oracle and LT, which also avoid searching on the graph. We also utilize the hierarchical structure to compute kNN results.

B. Network Representation Learning

There are many approaches to learn low-dimensional network representations, which embed the "social information" of each node, and then use these vectors for addressing machine learning tasks. DeepWalk [23] uses cosine distance between embedding vectors to approximate $p(v_j|v_i)$ representing the similarity of two nodes and use random walk statistics to learn the embedding. LINE [29] distinguishes the similarity of two nodes into "first-order" (directly close) and "second-order" (having similar neighborhoods) and uses cosine distance based formulas to represent them. Besides, there is a hierarchical training strategy called HARP [8] to improve the training efficiency and quality.

These methods capture the *social relations* of nodes, *e.g.,* neighborhood similarity and community membership, which is used for classification tasks, *i.e.,* "whether similar". Different from that, we capture the *distance relations* of vertices and focus on a regression task, *i.e.,* "how far". Note that the social embedding is insufficient for distance representation, which is verified in experiments. It's challenging to design such an embedding model with a well-suited representation metric that can capture the properties of road networks, which measures the *distance* instead of *similarity*.

III. Road Network Embedding Model

In this section, we first state the problem (Section III-A) and then introduce our road network vertices embedding model (Section III-B and Section III-C) and discuss how to train it (Section III-D). Table I summarizes the notations.

A. Problem Formalization

Road Networks. A *road network* is modeled as a weighted graph $G = (V, E, W)$, where road joints and road segments are modeled as graph vertices and edges respectively, with each edge associated with a positive edge weight denoting the road's length. Specifically, $v_i \in V$ is a *vertex*, $e_{v_i v_j} \in E$ is an *edge* from v_i to v_j, and $\omega_{v_i v_j} \in W$ is the *weight* of $e_{v_i v_j}$.

Shortest-path. A *path* is a sequence of adjacent edges between two vertices. We use $\rho_{v_s v_t} = \langle e_{v_1 v_2}, e_{v_2 v_3} \ldots, e_{v_{l-1} v_l} \rangle$, where $v_1 = v_s, v_l = v_t$ and $e_{v_i v_{i+1}} \in E$ for $i \in [1, l)$ to denote a *path* from v_s to v_t. Among all such paths, $\rho^*_{v_s v_t}$ is a shortest one, which captures how to traverse from v_s to v_t on a road network, and is denoted as a *shortest-path*.

Given a road network $G = (V, E, W)$ and a vertex pair (v_s, v_t) as a query, we need to answer the *shortest-path distance* denoted as

$$\phi(v_s, v_t) = \phi(\rho^*_{v_s v_t}) = \sum\nolimits_{e_{v_i v_j} \in \rho^*_{v_s v_t}} \omega_{v_i v_j}$$

Example 1 (Road Network.): The left part of Figure 1 shows a road network with 13 vertices and 15 edges, where each edge is associated with a positive weight in both directions, *e.g.,* $w_{v_4 v_3} = w_{v_3 v_4} = 2$. Among paths from v_4 to v_8, $\rho_2 = \langle e_{v_4, v_3}, e_{v_3, v_6}, e_{v_6, v_8} \rangle$ is the shortest one with distance of 8.

B. Vertices Embedding Model

Intuitively, we can use a $|V| \times |V|$ matrix to index all pairs of shortest distances between vertices. However, it is impractical as the space cost $\Theta(|V|^2)$ can be very large (*e.g.,* for road networks with millions of vertices). Therefore, we propose to embed all vertices in a road network into a d-dimensional space $\mathbb{R}^d (d << |V|)$ and utilize the embedding with some metric (*e.g.,* distance of vectors) or function (*e.g.,* represented by a neural network) to represent the shortest-path distances. We use the L_p-distance as the metric, with an acceptable error rate (*e.g.,* less than 1%) and we will explain why we choose it in Section III-C.

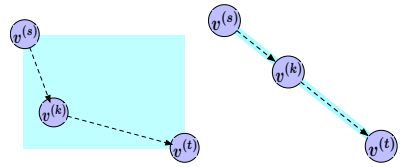
Road Network Vertex Embedding. Given a road network $G = (V, E, W)$, we embed each vertex $v_i \in V$ into a d-dimensional space as a vector $\boldsymbol{v}^{(i)}$ and we denote the embedding matrix as $M = \{\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \cdots, \boldsymbol{v}^{(n)}\}$. For a query from v_s to v_t, we use the *L_p-distance* of the vector pair to approximate the shortest-path distance denoted by

$$\hat{\phi}(v_s, v_t) = ||\boldsymbol{v}^{(s)} - \boldsymbol{v}^{(t)}||_p = \big|\big|M[s] - M[t]\big|\big|_p$$

We evaluate the performance by using the approximation rate between $\hat{\phi}(v_s, v_t)$ and $\phi(v_s, v_t)$, which is measured by *absolute error* (e_{abs}) and *relative error* (e_{rel}).

$$e_{abs}(v_s, v_t) = |\hat{\phi}(v_s, v_t) - \phi(v_s, v_t)|$$
$$e_{rel}(v_s, v_t) = |\hat{\phi}(v_s, v_t) - \phi(v_s, v_t)| \ / \ \phi(v_s, v_t)$$

Example 2 (RNE): The right part of Figure 1 shows an example of road network embedding. We utilize the embedding vector pair $\boldsymbol{v}^{(1)}$ and $\boldsymbol{v}^{(8)}$ with L_p-distance to approximate the shortest-path distance $\phi(v_1, v_8)$ as $\hat{\phi}(v_1, v_8) = 8$, with $e_{abs} = e_{rel} = 0$;



(a) L_1 (b) $L_p(p > 1)$

Fig. 2: Distance-embedding freedom of $\boldsymbol{v}_k$ in $\mathbb{R}^2$

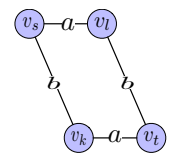


Fig. 3: A case where $L_p(p > 1)$ fails

C. Why L_p-Distance and Why Choosing $p = 1$

It is crucial to choose a representation metric to capture shortest-path distances of road networks. We use L_p-distance as the representation metric, which has the following salient properties.

1) Non-negativity. $||\boldsymbol{v}^{(s)} - \boldsymbol{v}^{(t)}||_p \geq 0$ (equality holds *iff* $\boldsymbol{v}^{(s)} = \boldsymbol{v}^{(t)}$).
2) Symmetry. $||\boldsymbol{v}^{(s)} - \boldsymbol{v}^{(t)}||_p = ||\boldsymbol{v}^{(t)} - \boldsymbol{v}^{(s)}||_p$.
3) Triangle Inequality. $||\boldsymbol{v}^{(s)} - \boldsymbol{v}^{(k)}||_p + ||\boldsymbol{v}^{(k)} - \boldsymbol{v}^{(t)}||_p \geq ||\boldsymbol{v}^{(s)} - \boldsymbol{v}^{(t)}||_p$.

The road-network distance between two vertices also satisfies these properties. Thus L_p-distance (instead of some representation functions, *e.g.,* approximated by a neural network) can be utilized to represent the road-network distance.

Although for L_p where $p > 1$ (*e.g.,* L_2, *i.e.,* Euclidean distance), the above properties still hold, we prefer L_1 because of the linearity. Intuitively, the representation ability of L_p-distance can be shown by its embedding freedom. Suppose we train an RNE for a road network where v_k is an intermediate vertex on the shortest path from v_s to v_t. As the shortest distance has $\phi(v_s, v_t) = \phi(v_s, v_k) + \phi(v_k, v_t)$ (equality in (3) holds), we have to adjust $\boldsymbol{v}^{(k)}$ such that $||\boldsymbol{v}^{(s)} - \boldsymbol{v}^{(t)}||_p = ||\boldsymbol{v}^{(s)} - \boldsymbol{v}^{(k)}||_p + ||\boldsymbol{v}^{(k)} - \boldsymbol{v}^{(t)}||_p$. This leads to the following constraints that need to satisfy:

- $p = 1$: $(\boldsymbol{v}^{(s)} - \boldsymbol{v}^{(k)})(\boldsymbol{v}^{(k)} - \boldsymbol{v}^{(t)}) \geq \mathbf{0}$, *i.e.,* for each dimension the coordinate of $\boldsymbol{v}^{(k)}$ is between $\boldsymbol{v}^{(s)}$ and $\boldsymbol{v}^{(t)}$. (*linearity*)
- $p > 1$: $\boldsymbol{v}^{(k)} = \lambda \boldsymbol{v}^{(s)} + (1 - \lambda) \boldsymbol{v}^{(t)}$ for some $\lambda \in [0, 1]$, *i.e.,* $\boldsymbol{v}^{(k)}$ must be on the line segment between $\boldsymbol{v}^{(s)}$ and $\boldsymbol{v}^{(t)}$.

Figure 2 shows the freedom space of $\boldsymbol{v}^{(k)}$ for L_1 and $L_p(p > 1)$ when $d = 2$. With a bigger d, $\boldsymbol{v}^{(k)}$ for L_1 has larger freedom space but that is not true for $L_p(p > 1)$. Theoretically, this is because L_1 has stronger representation ability than $L_p(p > 1)$ in embedding shortest-paths of a planar graph [3], [7], [9], [15], [18], [19], [24]. *e.g.,* Manhattan distance (L_1) can be utilized to roughly estimate the road distances, due to the planar and grid-like nature of road networks.

Example 3 (Superiority of L_1): Figure 3 shows a common case where both v_k, v_l are on the shortest path from v_s to v_t. Considering $L_p(p > 1)$, both $\boldsymbol{v}^{(k)}, \boldsymbol{v}^{(l)}$ have the form $\lambda \boldsymbol{v}^{(s)} + (1 - \lambda) \boldsymbol{v}^{(t)}$, which fails in making both $\hat{\phi}(v_k, v_l) = \phi(v_k, v_l)$ and $\hat{\phi}(v_s, v_t) = \phi(v_s, v_t)$, whatever the embedding dimension d we choose. But for L_1, we can simply let $\boldsymbol{v}^{(s)}, \boldsymbol{v}^{(t)}, \boldsymbol{v}^{(k)}$ and $\boldsymbol{v}^{(l)}$ be $(0, 0), (a, b), (0, b)$ and $(a, 0)$ respectively to fit the road network with $d = 2$.

Hence, L_1 is more compatible for representing shortest dis-

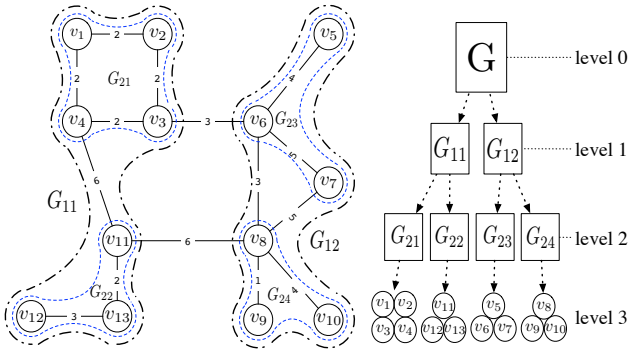


Fig. 4: Hierarchical graph partitioning

tances in road networks and we choose it as the representation metric of RNE. We will also show the superiority of L_1 metric in Section VII. Moreover, we will show that we can leverage the properties of L_1-distance to support kNN queries and range queries in Section VI.

D. RNE Training

The vertex-based RNE model will generate a $|V| \times d$ matrix $M = \{\boldsymbol{v}^{(i)}\}$. For each vertex pair (v_s, v_t), we aim to minimize $|\hat{\phi}(v_s, v_t) - \phi(v_s, v_t)|$. To this end, we first select a training dataset consisting of shortest distances of vertex pairs $S = \{(v_s, v_t, \phi_{(v_s,v_t)})\}$ and then minimize the square approximating error, i.e.,

$$\underset{M=\{\boldsymbol{v}^{(i)}\}}{\arg\min} \sum_{(v_s,v_t,\phi(v_s,v_t))\in S} \left| ||\boldsymbol{v}^{(s)} - \boldsymbol{v}^{(t)}||_1 - \phi_{(v_s,v_t)} \right|^2 \quad (1)$$

Function Training($M = \{\boldsymbol{v}^{(i)}\}$, α, S)

foreach $(v_s, v_t, \phi_{(v_s,v_t)}) \in S$ **do**
  $\mathcal{L} = \left(||M[s] - M[t]||_1 - \phi(v_s, v_t)\right)^2$ // loss
  $M[s] = M[s] - \alpha * \frac{\partial \mathcal{L}}{\partial M[s]}$ // α:learning rate
  $M[t] = M[t] - \alpha * \frac{\partial \mathcal{L}}{\partial M[t]}$

Intuitively, we can randomly select a set of vertex pairs, and learn the embedding matrix, as Function Training shows, *e.g.,* utilizing the stochastic gradient descent (SGD) to minimize the loss. However, this method has several limitations. First, because the vector space is too sparse, if we directly train the model, it is hard to converge, *i.e.,* many vertex embeddings can not be fitted into proper relative positions in $\mathbb{R}^d$, even when extra training epochs have been taken. Second, the training samples should be judiciously selected, because $|V|$ vertices spread all over the road network and it is hard to train all $|V|^2$ pairs that fit the global layout of all vertices on the road network. These issues significantly influence the query performance, so we propose a hierarchical embedding model in Section IV and select training samples in Section V.

IV. Hierarchical RNE Model

The basic vertex RNE is a shallow embedding model, *i.e.,* embedding lookup, which is in lack of parameter sharing

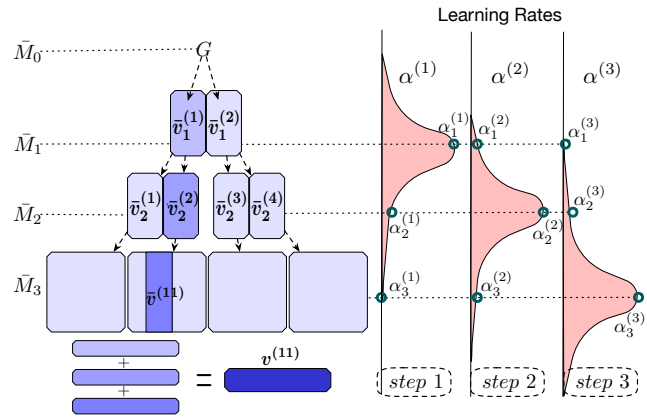


Fig. 5: Hierarchical local embedding model and training

and generalization. It's hard to train the $|V| \cdot d$ parameters to fit the $|V|^2$ distance relations between vertices. Thus the challenge is how to design an embedding model to incorporate the structure information of road networks. We propose a hierarchical RNE model (Section IV-A) and presents how to train it (Section IV-B).

A. Hierarchical Embedding Model

1) Graph Partitioning Hierarchy: To exploit the inherent hierarchy of road networks, we iteratively partition the road network graph into sub-graphs and construct a tree structure to capture the structure information.

Graph Partitioning. Given a road network $G = (V, E, W)$, a graph partitioning generates a set of sub-partitions $\{G_i = (V_i, E_i, W_i) | i \in [1, \kappa]\}$, where (1) $\bigcup_i V_i = V$ and $V_i \cap V_j = \emptyset$ if $i \neq j$; (2) $E_i = \{e_{v_x v_y} | v_x, v_y \in V_i\}$; (3) $W_i = \{w_e | e \in E_i\}$; (4) $\kappa > 1$ is the partitioning fanout. Graph partitioning aims to minimize the cut edges while making different sub-partitions have similar sizes.

Since graph partitioning minimizes the cut edges between sub-graphs, vertices within the same sub-graph have larger proximity than vertices from different sub-graphs. Hence graph partitioning captures the structure information of a road network. Then we get a road network partitioning hierarchy by recursively conducting graph partitioning on the road network, with a vertex set size threshold $\delta > 1$ (Figure 4 shows an example). The optimal graph partitioning problem has been proven to be NP-hard [17], and we adopt a famous multi-phase graph partition algorithm [17] to implement graph partitioning in each level.

Example 4 (Graph Partitioning Hierarchy): As Figure 4 shows, by partitioning road network G in Figure 1, we get sub-partitions $\{G_{11}, G_{12}\}$. Recursively partitioning G_{11} and G_{12}, we get $\{G_{21}, G_{22}\}$ and $\{G_{23}, G_{24}\}$, respectively. $\langle \{G\}, \{G_{11}, G_{12}\}, \{G_{21}, G_{22}, G_{23}, G_{24}\}, V \rangle$ forms a partitioning hierarchy of 4 levels.

2) Hierarchical Model: We utilize the tree structure to construct the hierarchical embedding model.

Hierarchical Road Network Embedding. Each tree node (sub-graph or real vertex) $v_{l,i}$ in level l of the hierarchy has a

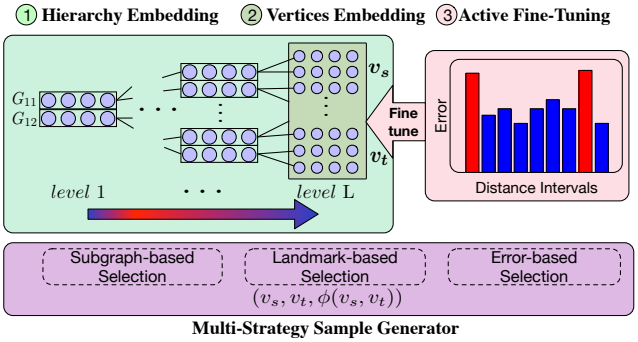


Fig. 6: Road Network Embedding Framework

local embedding $\bar{\boldsymbol{v}}_l^{(i)}$ representing the relative position among its siblings, forming the local embedding matrix of each level $\bar{M}_l = \{\bar{\boldsymbol{v}}_l^{(1)}, \bar{\boldsymbol{v}}_l^{(2)}, \cdots, \bar{\boldsymbol{v}}_l^{(n_l)}\}$. The *global embedding* of a node is the sum of its ancestors' local embeddings (including itself), *i.e.,* $\boldsymbol{v}^{(i)} = \sum_{v_{l,j} \in anc(v_i)} \bar{\boldsymbol{v}}_l^{(j)} = \sum_{v_{l,j} \in anc(v_i)} \bar{M}_l[j]$.

Example 5 (Hierarchical RNE): The left part of Figure 5 gives a hierarchical road network embedding including local embeddings $\bar{M}_1, \bar{M}_2, \bar{M}_3$. The global embedding of $\boldsymbol{v}^{(11)}$ is the sum of the local embeddings of its ancestors, *i.e.,* $\boldsymbol{v}^{(11)} = \bar{\boldsymbol{v}}_1^{(1)} + \bar{\boldsymbol{v}}_2^{(2)} + \bar{\boldsymbol{v}}_3^{(11)} = \bar{M}_1[1] + \bar{M}_2[2] + \bar{M}_3[11]$.

Although the number of embedding vectors of hierarchical embedding model ($|V| + \frac{|V|}{\delta(\kappa-1)}$) are larger than that of vertex embedding model ($|V|$), the sum of L_p-norms of embedding matrices ($\sum_l ||\bar{M}_l||_p$) is smaller than $||M||_p$. That's because the norms in higher level (with less nodes) are usually larger and play a decisive role in the global embedding of vertices in it, *e.g.,* for $||\boldsymbol{v}^{(11)}||_p$, $||\bar{\boldsymbol{v}}_1^{(1)}||_p \gg ||\bar{\boldsymbol{v}}_2^{(2)}||_p \gg ||\bar{\boldsymbol{v}}_3^{(11)}||_p$. But $||\bar{\boldsymbol{v}}_1^{(1)}||_p$ is counted only once for all its descendants. Therefore, the hierarchical RNE model shares parameters by leveraging a tree structure and the sum of norms of parameters is much smaller than the vertex RNE model. As a result, the learning phase converges much faster and the performance is much better, which we will show in Section IV-B.

B. Hierarchical Model Training

1) Hierarchical Learning Framework: The framework to learn a hierarchical RNE model is shown in Figure 6 and Algorithm 1. Given a road network $G = (V, E, W)$, we first construct a tree structure (Figure 4) and we will learn a local embedding hierarchy of L levels (Figure 5) in 3 phases.

① We first progressively train the whole hierarchical local embedding, *i.e.,* make it converge in a top-down manner.

② Next we fix the local embeddings of sub-graphs in the hierarchy and train the local embedding (vertices).

③ Then we fine-tune some under-fitting vertex embeddings through an active training sample selection strategy.

Note that we adopt different training sample selection strategies in different training phases, which will be introduced in Section V. Finally we transform the hierarchical local embedding $\{\bar{M}_l = \{\bar{\boldsymbol{v}}_l^{(i)}\}\}$ to a global vertex embedding matrix $M = \{\boldsymbol{v}^{(i)}\}$. For any shortest distance query (v_s, v_t),

Algorithm 1: Hierarchical Road Network Embedding

Input: $G(V, E, W)$: road network,
d: #dimensions, L: #partitioning levels
Output: M: RNE matrix

// ① Hierarchy embedding
Initialize $\bar{M}_l$ for $l = 1, \cdots, L$
for $lev = 1, 2, ..., L$ **do**
  $S^{(lev)}$ = *Subgraph-Level-Samples*$(\{G_{lev}\})$
  for $l = 1, 2, ..., L$ **do**
    $\alpha_l^{(lev)}$ = *Level-LearningRate*(lev, l)
  $\{\bar{M}_l\}$ = *TrainingHier*$(\{\bar{M}_l\}, \{\alpha_l^{(lev)}\}, S^{(lev)})$
// ② Vertices embedding
S = *Landmark-Based-Samples*(G)
$\{\bar{M}_l\}$ = *TrainingHier*$(\{\bar{M}_l\}, \{0, \cdots, 0, \alpha_L\}, S)$
// ③ Fine-Tuning
for $k = 1, 2, \cdots$ **do**
  S_k = *Error-Based-Samples*$(\{\bar{M}_l\})$
  $\{\bar{M}_l\}$ = *TrainingHier*$(\{\bar{M}_l\}, \{0, \cdots, 0, \alpha_L\}, S_k)$
// Transform to global vertices embedding
foreach $v_i \in V$ **do**
  $M[i] = \sum_{v_{l,j} \in anc(v_i)} \bar{M}_l[j]$
return M

Function TrainingHier($\{M_l\}$, $\{\alpha_l\}$, S)

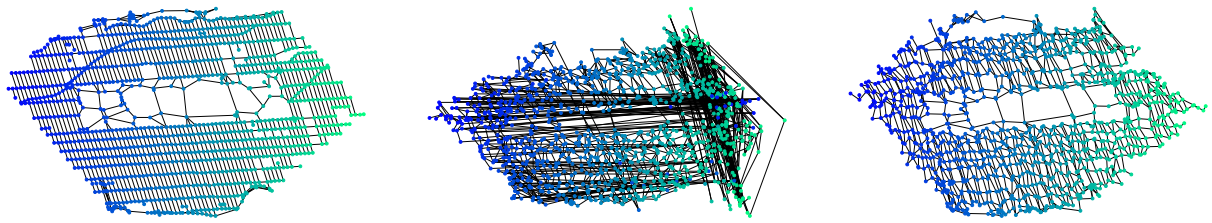
foreach $(v_s, v_t, \phi) \in S$ **do**
  $\hat{\phi} = \left|\left| \sum_{v_{l,j_s} \in ans(v_s)} \bar{M}_l[j] - \sum_{v_{l,j_t} \in ans(v_t)} \bar{M}_l[j_t] \right|\right|_1$
  $\mathcal{L} = (\hat{\phi} - \phi)^2$ // loss
  foreach $v_{l,j_s} \in ans(v_s)$ **do**
    $\bar{M}_l[j_s] = \bar{M}_l[j_s] - \alpha_l * \frac{\partial \mathcal{L}}{\partial \bar{M}_l[j_s]}$
  foreach $v_{l,j_t} \in ans(v_t)$ **do**
    $\bar{M}_l[j_t] = \bar{M}_l[j_t] - \alpha_l * \frac{\partial \mathcal{L}}{\partial \bar{M}_l[j_t]}$
return $\{\bar{M}_l\}$

we use the embedding vectors to calculate $\hat{\phi}(v_s, v_t) = ||\boldsymbol{v}^{(s)} - \boldsymbol{v}^{(t)}||_1 = \left|\left|M[s] - M[t]\right|\right|_1$ to get an approximate result.

2) Learning Procedure: For phase ①, based on the partitioning hierarchy of a road network, we will learn the hierarchical local embedding matrix $\{\bar{M}_l = \{\bar{\boldsymbol{v}}_l^{(i)}\}\}$ by minimizing

$$\underset{\{\bar{\boldsymbol{v}}_l^{(i)}\}}{\arg\min} \sum_{\substack{(v_s, v_t, \phi(v_s, v_t)) \\ \in S}} \left| ||\sum_{\substack{v_{l,j_s} \in \\ anc(v_s)}} \bar{\boldsymbol{v}}_l^{(j_s)} - \sum_{\substack{v_{l,j_t} \in \\ anc(v_t)}} \bar{\boldsymbol{v}}_l^{(j_t)}||_1 - \phi_{(v_s, v_t)} \right|^2 \tag{2}$$

The hierarchical embedding training process can also be viewed in Function TrainingHier. A question is how to learn the embeddings of sub-graphs in different levels, which are of different norm scales, *i.e.,* the importance of the embedding in



(a) Original network (b) Straight (c) Hierarchical

Fig. 7: Comparison between embeddings trained straightly and trained hierarchically

a higher level is higher than a lower one. Intuitively, we choose a learning rate α_l for each level l, and update $\bar{M}_l$ by every training sample. However, embedding in each level cannot converge if trained simultaneously, *e.g.,* the update of embeddings in higher levels may cause the embeddings in lower levels to be adapted, which leads to a poor model performance. Therefore, we propose to learn the embedding hierarchy level-by-level (totally L steps), *i.e.,* first coarse-grained embeddings (large sub-graphs) then fine-grained embeddings (smaller sub-graphs). We achieve that by adjusting the learning rates after each step. *e.g.,* the learning rate of the embedding of a higher level is set large in early steps and progressively decays, and that of a lower level are the opposite (right part of Figure 5 shows).

As shown in lines 2–6 in Algorithm 1, we learn the local embeddings hierarchy through L steps, focusing on levels from top to down respectively. In each step, we focus on level lev and choose samples for that level (line 3). Then we choose learning rates $\alpha_l^{(lev)}$ for $l = 1, \cdots, L$ (line 5) where a level l near to lev has a bigger learning rate, *e.g.,* $\alpha_l^{(lev)} = \frac{\alpha_0}{|l-lev|+1}$ with an α_0. Then we update the local embeddings by the corresponding learning rates using SGD (line 6) and move to the next step.

The local embedding hierarchy gives representations of all sub-graphs, and now we focus on learning $\bar{M}_L$ for all real vertices in the hierarchy (phase ②), representing the bias of each v_i to its associated leaf sub-graph and its norm $\left|\left|\bar{M}_L\right|\right|_p$ is usually small. As shown in lines 7–8, we first carefully select appropriate training samples and then fix $\{\bar{M}_l | l < L\}$ by setting α_l to 0 and only update the vertex local embedding with learning rate α_L.

Example 6 (Road Network Hierarchy Embedding): The embedding process of the hierarchical local embedding in the left part of Figure 5 is shown in the right part of it. We first initialize $\bar{M}_1, \bar{M}_2, \bar{M}_3$ randomly (line 1). Then we train them through distances of vertex pairs (*e.g.,* $(v_4, v_8, 8)$ where $\boldsymbol{v_4} = \bar{M}_1[1] + \bar{M}_2[1] + \bar{M}_3[4]$ and $\boldsymbol{v_8} = \bar{M}_1[2] + \bar{M}_2[4] + \bar{M}_3[8]$ as a training sample) (lines3). In step 1, we use learning rates $\alpha^{(1)}$ in which $\bar{M}_1$ takes the biggest one $\alpha_1^{(1)}$ (line 5) to train the embeddings (line 6). While in steps 2 and 3, $\bar{M}_2$ and $\bar{M}_3$ take the biggest learning rates among $\alpha^{(2)}$ and $\alpha^{(3)}$ respectively.

Example 7 (Vertices Embedding): We fix the local embeddings $\bar{M}_1, \bar{M}_2$ ($\alpha_1 = \alpha_2 = 0$) and train $\bar{M}_3$ with α_3 (line 8).

It is worth noting that based on the hierarchical embedding strategy, the training of RNE converges much faster than the basic training of a flat vertices embedding, and also gains a better performance for shortest-path distance representation, because we consider the global road network layout and the correlation among vertices. In the partition hierarchy, the number of coarse-grained sub-graphs in upper levels (suppose the set of sub-graphs in level l is P_l) are small and grows gradually as the level increases. Therefore we can easily level-by-level embed the relative positions of the sub-graphs in each level (*i.e.,* only $|P_l|^2$ pairs), with the representations of upper-level sub-graphs serving as good initializations for those of lower levels. So we use less time to gain a higher-quality RNE, which well preserves the global structure of the whole vertex set.

Example 8 (A real-world RNE example.): Figure 7 shows a 2-dimensional RNE of Manhattan Island in New York City, where Figure 7a is the original road network, Figure 7b is the vectors trained without hierarchical structure, and Figure 7c is the vectors trained hierarchically. As we can see, without a hierarchical road network structure, there are many embedding vectors collapse to small groups in the corner. And with a hierarchical structure, our method gets well distributed embedding vectors for vertices, which preserve the global layout of the original road network.

V. Training Sample Selection

Intuitively, we want to use all vertex pairs ($|V|^2$) from the road network $G = (V, E, W)$ to train the embeddings. However, more training data may not bring better embedding performance, as predictive information grows slower than the amount of Shannon entropy [6]. Besides, huge training data may cause over-fitting in unpredictable data distribution. Thus, we need to select an appropriate number of training samples, which can properly represent the road network and also meet the capacity of our model.

A. Samples for Hierarchy Embedding

The goal is to select high-quality training samples for embedding sub-graphs in a given level l. The naive random selection strategy chooses different vertex pairs uniformly, but we select sub-graph pairs uniformly, *i.e.,* choose vertex pairs uniformly from different sub-graph pairs. Therefore, the embedding can accurately represent the sub-graphs of that level, *i.e.,* the relative positions of $|P_l|^2$ pairs.

Algorithm 2: Training Sample Selection

Input: $G = (V, E, W)$: Road network,
L: #levels, N: #samples,
P_l: The set of sub-graphs on l-th level,
U: The set of landmarks,
I_i: The set of vertex pairs in the i-th interval
Output: $S = \{(v_s, v_t, \phi_{(v_s,v_t)})\}$: N training samples

// ① Subgraph-level sample selection
1 **for** $l = 1, 2, \ldots, L$ **do**
2 **for** $k = 1, 2, \ldots, N$ **do**
3 $G_{li}, G_{lj} = UniformSelection(P_l, P_l)$
4 $v_s, v_t = UniformSelection(V_{li}, V_{lj})$
5 $S_l.add((v_s, v_t, \phi_{(v_s,v_t)}))$

// ② Landmark-based sample selection
6 **for** $k = 1, 2, ..., N$ **do**
7 $v_s, v_t = UniformSelection(U, V)$
8 $S.add((v_s, v_t, \phi_{(v_s,v_t)}))$

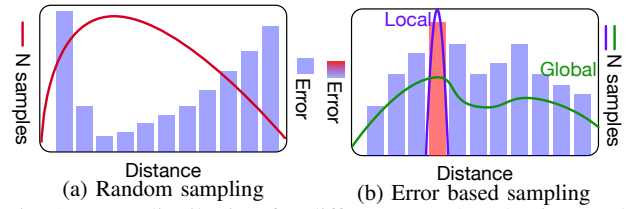
// ③ Error-based sample selection
9 $\{e_i\} = GetErrors(\{I_i\})$
10 **for** $k = 1, 2, ..., N$ **do**
11 **if** *flag* = *local* **then**
12 $\theta = \underset{i}{\operatorname{argmax}} \{e_i\}$
13 $I = I_\theta$
14 **if** *flag* = *global* **then**
15 $I = WeightedSelection(\{I_i\}, \{e_i\})$
16 $(v_s, v_t) = UniformSelection(I)$
17 $S.add((v_s, v_t, \phi_{(v_s,v_t)}))$

Specifically, as shown in lines 1–5 in Algorithm 2, suppose the set of sub-graphs in level l is $P_l = \{G_{li}\}$, we first randomly choose a sub-graph pair $G_{li}, G_{lj} \in P_l$ with the probability $\frac{1}{|P_l|^2}$, then randomly choose two vertices from G_{li} and G_{lj} respectively. So we have a vertex pair (v_s, v_t) where $v_s \in V_{li}, v_t \in V_{lj}$ are chosen with the probability $\frac{1}{|P_l|^2 * |V_{li}| * |V_{lj}|}$ depending on the sizes of the associated sub-graph pair.

B. Samples for Vertices Embedding

The goal is to select high-quality training samples for embedding vertices. We update the vertex embedding to capture the relative positions of vertices among the associated leaf-node. However, if we randomly select samples, an embedding vector has conflicts being updated concurrently by many different samples (*i.e.,* different peer vertices), which leads to poor model performance. Therefore, we propose a landmark-based sample selection strategy to overcome the problem.

Landmark based sample selection. There are usually some"pivot" vertices which are more representative for the whole network structure and we call them landmarks. We take them as the reference points for all the other vertices. Intuitively, with landmark-based sample selection strategy we can control the relative embedding positions based on embeddings of important vertices.



(a) Random sampling (b) Error based sampling

Fig. 8: Error distribution for different sample selection methods

As shown in lines 6–8 in Algorithm 2, we select $U \subset V$ as the landmark set where $|U| \ll |V|$, and choose training samples from $\{(v_s, v_t, \phi_{(v_s,v_t)}) | v_s \in U, v_t \in V\}$, *i.e.,* each training sample gives the relationship between a vertex and a landmark. Thus the probability to be chosen is $\frac{1}{|U|}$ for each landmark (since $|U| \ll |V|$) and $\frac{1}{|V|}$ for others. If we train the model with N samples, the expected number of each landmark to be included is enough ($\frac{N}{|U|} \gg \frac{N}{|V|}$), which arranges the $|U|$ embedding vectors to the suitable positions quickly. Then other embedding vectors are trained according to their distances to the $|U|$ landmarks and converge better than when there are no clear references. Moreover, since the sample space for the landmark-based strategy ($|U| \cdot |V|$) is much smaller than the naive strategy ($|V|^2$), it can reduce the over-fitting problem. Thus the landmark-based strategy leads to a better performance.

Landmark Selection. There are many proposed methods to carefully choose landmarks [13], [14], which lead to good performance. For example, *farthest selection* iteratively selects the farthest vertex to existing selected landmarks, that tries to identify regions of the graph that are not "well-covered" by avoiding existing landmarks.

C. Active Fine-Tuning

As Figure 8a shows, the distances of randomly chosen samples do not distribute evenly but concentrate in a narrow distance range (curve). Consequently, the vertex embedding may get large errors in those distance ranges (tall bars), thus the model incurs high total error rate and large error variance. Therefore, we propose an active distance-based sample selection strategy to fine-tune the model. The main idea is to choose the high-demand samples (*e.g.,* samples from under-fitting intervals) according to the current error distribution and dynamically adjust the sample selection:

1) First we divide vertex pairs into R distance intervals (or buckets) with equal lengths.
2) Next we generate R sets on the R intervals to reflect the error-distance distribution of the current model.
3) Then we iteratively fine-tune the embedding of vertex pairs in intervals with the highest approximation errors by selecting more training samples on them.

Distance-based sample selection. Note that for Step 1, a problem is how to randomly choose samples from each interval I_i, as we cannot maintain all $|V|^2$ vertex pairs by R distance intervals due to the time and space limitation. An alternative

choice is to partition the whole road network into grids and to maintain all the grid pairs by distance as an approximation of the distances of vertex pairs inside each grids.

Suppose we partition the road network into $K \times K$ grids based on the spatial positions of vertices, where grid i ($1 \leq i \leq K^2$) contains the vertex set g_i. We define the *distance* of a grid pair (g_s, g_t) as *the least number of grids we need to go across* from g_s to g_t, which always falls in $[0, 2K-2]$ (0 when $g_s = g_t$ and $2K-2$ when g_s and g_t are two ends of a diagonal), based on which we classify the $K^2 \times K^2$ grid pairs into $2K-1$ buckets $\{\hat{I}_i | i = 0, \cdots, 2K-2\}$. As the partition is equidistant, we can set $R = 2K-1$, and use grid pair buckets $\{\hat{I}_0, \cdots \hat{I}_{2K-2}\}$ to approximate vertex pair buckets $\{I_0, \cdots I_{R-1}\}$.

Concretely, we first choose a grid pair $(g_s, g_t) \in \hat{I}_i$ (with probability $\frac{|g_s| \cdot |g_t|}{\sum_{g_j, g_k \in \hat{I}_i} |g_j| \cdot |g_k|} = \frac{|g_s| \cdot |g_t|}{|\hat{I}_i|}$) of a chosen distance interval i (with probability $\frac{1}{2K-1}$), then we randomly choose one vertex from g_s and g_t respectively (with probability $\frac{1}{|g_s|}$ and $\frac{1}{|g_t|}$) to get a vertex pair. In this way, we choose samples with equal probabilities ($\frac{1}{(2K-1) \cdot |\hat{I}_i|} \approx \frac{1}{R * |I_i|}$) among all the pairs in each $\hat{I}_i$, and only $O(K^4)$ extra space for storage and $O(1)$ time to retrieve a sample by using a hash table.

Error-based samples selection. For Step(3), in each turn, we use last round's validation statistics, *e.g.,* e_{rel} and $var(e_{rel})$, to evaluate the model performance on each interval and try to choose more samples in the intervals with high errors. As the two curves in Figure 8b show, we can either choose pairs only from the interval bucket with the highest error (*Local*) or assign samples to every bucket with the amount proportional to its error (*Global*).

As the approximation errors of vertex pairs in "bad" intervals evidently decrease, the whole error expectation and variation decreases and the estimation stability improves.

VI. Range and kNN Queries

In the processing of k-nearest neighbor (kNN) queries, it's crucial how to utilize indexes that leverage the structures of road networks to reduce the searching space, *i.e.,* prune the unpromising expansions. We discuss how to utilize our hierarchical embedding model to support efficient range queries and kNN queries.

Range Queries. A range query takes as input a source vertex v_s, a set V_T of target vertices, and a threshold τ, and outputs all vertices in V_T whose L_p-distances to $\boldsymbol{v}_s$ is within τ.

kNN Queries. A kNN query takes as input a source vertex v_s, a set V_T of target vertices and a number k, and outputs k vertices from V_T that have the smallest L_p-distances to $\boldsymbol{v}_s$.

Tree-Structured Index. We still use the tree structure (Figure 5) to build an index. For each tree node $node$, besides the global embedding vector of the node, we also maintain a radius $\mathbf{r}(node)$ which is the maximum L_p-distance between this global embedding vector and all global embedding vectors of vertices under this node, i.e.,

$$\mathbf{r}(node) = \max_{v' \in node} L_p(\boldsymbol{v}_{node}, \boldsymbol{v'}).$$

Range Algorithm. We traverse the tree from the root node. For each node $node$ with the radius of $\mathbf{r}(node)$, if

$$L_p(\boldsymbol{v}_s, \boldsymbol{v}_{node}) - \mathbf{r}(node) > \tau,$$

we can prune the node based on the triangle inequality; otherwise,

(1) if the node is a leaf, we enumerate the vertices in the node, compute the L_p-distances to $\boldsymbol{v}_s$, and add the vertices whose distances are not larger than τ into the result set;

(2) if the node is a non-leaf node, we recursively traverse or prune each of its children.

kNN Algorithm. We utilize the tree index to support kNN queries. We first maintain a min-priority queue Q which maintains a set of tuples $\langle node, distance \rangle$ and an answer list. Initially, we put tuple $\langle root, \max(L_p(\boldsymbol{v}_s, \boldsymbol{v}_{root}) - \mathbf{r}(root), 0) \rangle$ into Q. Next we repeat the following iterations until the Q is empty or we get k results. In each iteration, we pop the tuple with the smallest distance from the queue, *e.g.,* $\langle e, d \rangle$.

(1) If e is a non-leaf tree node, we enumerate each of its children e_c, and put $\langle e_c, \max(L_p(\boldsymbol{v}_s, \boldsymbol{v}_{e_c}) - \mathbf{r}(e_c), 0) \rangle$ into Q;

(2) If e is a leaf node, we enumerate each vertex e_v in the node and put $\langle v, L_p(\boldsymbol{v}_s, \boldsymbol{v}_{e_v}) \rangle$ into Q;

(3) If e is a vertex, we put e into the result set.

VII. Experiments

We conduct extensive experiments on real-world datasets to evaluate our method and compare with state-of-the-arts.

A. Datasets and Experiment Setting

TABLE II: Details of datasets

Dataset	Region	#Vertices	#Edges
BJ	Beijing	338,024	881,050
FLA	Florida	1,070,376	2,687,902
US-W	Western-USA	6,262,104	15,119,284

Datasets. We use 3 real-world road networks with different data scales: **BJ**, **FLA** and **US-W**. Table II provides details of the datasets.

Experiment Setting. The models proposed in this paper are trained under Python 3.6 and TensorFlow 1.10. To achieve high efficiency, the testing programs are implemented in C++ and compiled with GNU GCC 4.4. All experiments are conducted in a machine with 2.40 GHz Intel CPU E52630, 128 GB RAM, running Ubuntu 14.04.

B. Ablation Studies

In this section, we perform ablation experiments over a number of facets of our RNE model in order to validate the effectiveness of them. The experiments are all performed on the **BJ** dataset. Performances reflected by the relative errors (e_{rel}) are evaluated on a validation set consisting of 1M randomly chosen samples.

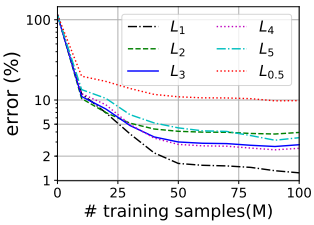


Fig. 9: Vary L_p metric

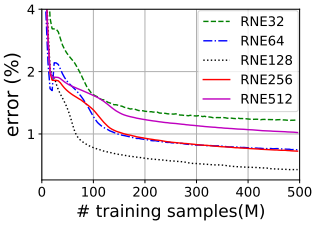


Fig. 10: Vary dimension d

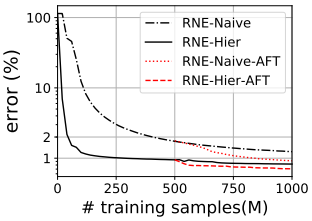


Fig. 11: Effect of hierarchical training and fine-tuning

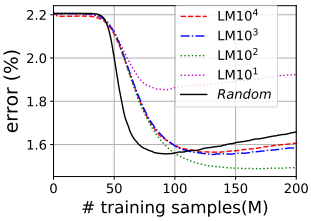


Fig. 12: Effect of landmark-based sample selection

1) The effect of representation function: As we discussed in Section III, L_p metric is a good choice as the function of our RNE to represent shortest-path distances while social embedding is insufficient for that. To verify this statement, we compare our RNE with L_1 metric and $d = 64$ against a baseline model with a social embedding and a neural network for distance regression:

- **DeepWalk Regression(DR)**: we train an embedding using DeepWalk [23] ($d = 64$) and take the vectors along with the longitudes and latitudes of the vertex pair as input $(\boldsymbol{v}^{(s)}, \boldsymbol{v}^{(t)})$. We then concatenate $[\boldsymbol{v}^{(s)}, \boldsymbol{v}^{(t)}, |\boldsymbol{v}^{(s)} - \boldsymbol{v}^{(t)}|]$ ($d = 3 \cdot (64 + 2)$) and use 3 different fully-connected neural networks with 1K, 10K, 100K parameters respectively for shortest distance regression. The 3 model are denoted as DR-1K, DR-10K, DR-100K, respectively.

We train the 4 models using training sets of different sizes for epochs until the validation errors stop decreasing and report the results in Figure 14. Two baselines that use longitudes and latitudes to roughly estimate the road network distances are also included: (1) the Manhattan distance; (2) the Euclidean distance.

We can observe that (1) DR significantly outperform Manhattan and Euclidean since it utilizes vectors which embed the similarity between nodes; (2) the error of RNE decreases more significantly with a larger training set (shown as their ratios to $|V|$) and is significantly lower than others when the training set has a reasonable size, *e.g.,* larger than $1 \cdot |V|$. This is because DR utilizes a pretrained network embedding but RNE doesn't, so RNE needs more data to fit. Nevertheless, RNE has stronger representation ability in approximating network distance, so its accuracy gets significantly lower with more data. On another side, the number of operations (*i.e.,* execution time) to take during inference of RNE (64) is extremely smaller than that of DR (1K, 10K, 100K). Therefore, RNE has the fastest query processing speed in the meanwhile achieving the lowest error rate. Thus L_1 is a compatible metric for network distance representation and RNE is an effective model for shortest-path approximation.

2) The effect of L_p: To verify the statement in Section III-C that L_1 norm is better than other $L_p (p > 1)$ as the RNE metric, we report the training results using the same training samples (100M), the same $d = 64$, and varying 6 L_p metrics, *i.e.,* $L_{0.5}$, L_1, L_2, L_3, L_4 and L_5. We can observe from Figure 9 that after convergence, L_1's error is significantly smaller than others (the errors do not show a consistent tendency with a bigger/smaller p). Thus we recommend to use L_1 metric to represent the shortest-path distances.

3) The effect of d: The embedding dimension d is a critical parameter in our model. To evaluate how d affects the performance, we train 5 models with $d = 32, 64, 128, 256, 512$. We can observe from Figure 10 that the mean errors for all 5 models decrease with more training samples but the decreasing speeds gradually slow down. mple, for RNE32, the model converges at 100M data of training. Theoretically, RNE512 can achieve the best representation ability with adequate samples but the training time is too long.

Considering the query time ($O(d)$) and the index size ($O(|V| \cdot d)$), a larger d does not always bring better performance. Based on the trade-off of model capacity (accuracy), training complexity and query efficiency, we choose the most suitable d for each road network as: **BJ** ($d = 64$), **FLA** ($d = 128$), and **US-W** ($d = 128$).

4) Effect of hierarchical embedding and fine-tuning: To evaluate how hierarchical embedding and active fine-tuning (See Section IV and Section V respectively) affect the training speed and model performance, we train the following models with $d = 64$:

1) **RNE-Naive**: simple vertex RNE model
2) **RNE-Hier**: hierarchical RNE model
3) **RNE-Naive-AFT**: RNE-Naive with active fine-tuning
4) **RNE-Hier-AFT**: RNE-Hier with active fine-tuning

Figure 11 shows the training errors of different models. As we can observe, for all 4 models, the errors decrease when more training samples are fed. Comparing to RNE-Naive, RNE-Hier can achieve a lower error with much less data, *e.g.,* error of RNE-Hier gets 1% by 250M samples while error of RNE-Naive is still around 1.5% by 750M. It shows that the hierarchical embedding can significantly speed-up the training process and gain a better performance.

However, the errors stop decreasing and go to converge for both RNE-Naive and RNE-Hier. Hence we apply the active fine-tuning process to make the errors decrease further more (red dashed). The result shows that our active sample selection strategy improves the accuracy by selecting samples on the under-fitting intervals.

5) The effect of landmark-based sample selection: As we discussed in Section V-B, we take the landmark-based sample selection strategy (LM) for vertices embedding. In this experiment, we randomly choose 4 different amounts of landmarks (*i.e.,* LM with $10^1, 10^2, 10^3, 10^4$). Then we choose pairs respectively based on the 4 landmark sets and choose pairs randomly (Random) as comparison. Thus we have 5

	Relative error (%)			Query time (us)		
	BJ	**FLA**	**US-W**	**BJ**	**FLA**	**US-W**
Euclidean	11.8	13.5	11.2	0.01		
Manhattan	16.0	13.5	16.0			
H2H	0			0.77	1.15	1.42
CH				10,665	22,173	102,558
Distance Oracle	5.32	-	-	1.58	-	-
ACH	3.73	3.99	3.85	2,029	8,576	33,385
LT	1.76	1.16	0.98	0.26	0.37	0.43
RNE	0.60	0.63	0.48	0.06	0.15	0.15

TABLE III: Mean error rate and query time

models all with $d = 64$, which use the same initialized embeddings gotten by road network hierarchy embedding.

Figure 12 shows the corresponding results on mean relative error during vertices embedding training. We can observe that the errors for 5 models first decrease as trained by more epochs and then increase because of over-fitting. The errors at the lowest points of the curves have: $\mathsf{LM}10^2 < \mathsf{Random} \approx \mathsf{LM}10^3 \approx \mathsf{LM}10^4 < \mathsf{LM}10^1$. As $\mathsf{LM}10^2$ outperforms Random, it's indicated that a limited landmark set of suitable size leads to less over-fitting and better performance. Meanwhile $\mathsf{LM}10^1$ performs worse than Random which indicates that too few landmarks cannot well represent the structure of the road network, which leads to bad accuracy. Thus, $\mathsf{LM}10^2$ has the best performance and we take it as the sample selection strategy for vertex embedding.

C. Comparison with State-of-the-arts

We compare our method RNE with other 5 state-of-the-art algorithms for shortest-path distance queries:

1) **CH**: Contraction Hierarchy [11]
2) **H2H**: Hierarchical 2-Hop Index [22]
3) **ACH**: Approximate CH [12]
4) **Distance Oracle**: [27]
5) **LT**: Distance estimation in ALT [13] based on landmarks and the triangle inequality

CH and H2H are 2 exact approaches and ACH, Distance Oracle and LT are 3 approximate methods. For ACH and Distance Oracle the parameter ϵ (upper bound of relative error) is adjustable. LT generates landmark set U to compute a $|U| \times |V|$ matrix and then utilize the triangle inequality to approximate the shortest distances, whose accuracy and speed are affected by $|U|$. For better experiment illustration, we only show ACH with $\epsilon = 0.1$. Due to the limitation of scalability of Distance Oracle, it can only work on **BJ** with ϵ up to 0.5, so we take that as the configuration of Distance Oracle here. And the best dimension parameters of RNE we chose for each road network are **BJ**-64, **FLA**-128, **US-W**-128. The sizes of landmark sets of LT we choose are **BJ**-128, **FLA**-256, **US-W**-256.

1) Query Efficiency: We shows the average query times of all the methods on each dataset in Table III. We also evaluate the query efficiency by varying different query distance scales. For each dataset, we generate Q groups of queries for various distance scales, where $Q = 5$ for small road networks **BJ**, $Q = 7$ for large road networks **FLA** and **US-W**. Each group consists of 10K random chosen queries and we test the average query time as shown in Figure 13 (the x-axis represents the upper bound of sample distance for each group). We make the following observations.

(1) For CH and ACH, the time cost of queries tends to increase as the distance gets larger. H2H also has the increasing tendency but not significantly. While the querying efficiencies of RNE and LT keep the same and that of Distance Oracle decreases. This is because for CH, when the distance increases, the number of hops in the shortest path ($|\rho^*|$) becomes larger. Therefore, the searching space for Dijkstra-like searching grows larger. For H2H, the time complexity is $O(w)$ where w is the tree-width and it grows sub-linearly with the number of vertices. Although the searching space gets larger, the querying time keeps stable. For Distance Oracle, the time complexity is $O(\log |V|)$ but the query is faster on larger distance scale, because Distance Oracle gets the distance of a vertex pair through looking up the block pair containing it, and the vertex pairs of larger distance scale are more likely to be in the same block pairs, where more memory access can hit in the cache, which makes queries faster. For LT and RNE, the computing times $O(|U|)$ and $O(d)$ fully depend on the size of the landmark set U and the embedding dimensions d, even for different datasets of different sizes.

(2) ACH is faster than CH since the searching stops earlier on the looser termination condition. Distance Oracle and H2H are $10^2 - 10^5$ faster than CH and ACH. LT is around 2x faster than H2H and Distance Oracle. RNE is around 2x faster than LT. Furthermore, LT and RNE can be paralleled on single query's level, and is theoretically able to speedup from $O(d)$ to $O(\log d)$ (*e.g.,* from 64 to 6) with SIMD and GPU.

2) Index Size and Building Time: In this experiment, we focus on the memory usage and index building time for every method. Table IV shows the results. Note that the index building time of RNE includes the time for generating samples and training embedding. We make the following observations. (1) The index size of each of the 4 algorithms increases as $|V|$ increases. (2) Distance Oracle has the highest space usage ($O(\frac{|V|}{\epsilon^2})$) (8GB for **BJ**). (3) H2H has the fastest growth rate (super-linearly with dataset size) , which may be impractical

	Index size (MB)			Building time (min)		
	BJ	**FLA**	**US-W**	**BJ**	**FLA**	**US-W**
H2H	827	1,748	19,696	< 1	< 1	6
CH	88	272	1,620	171	2,094	>100,000
Distance Oracle	8,457	-	-	3	-	-
ACH	54	153	976	35	566	19,282
LT	166	1,046	6,116	<1	5	38
RNE	83	523	3,058	34	66	113

TABLE IV: Index size and building time

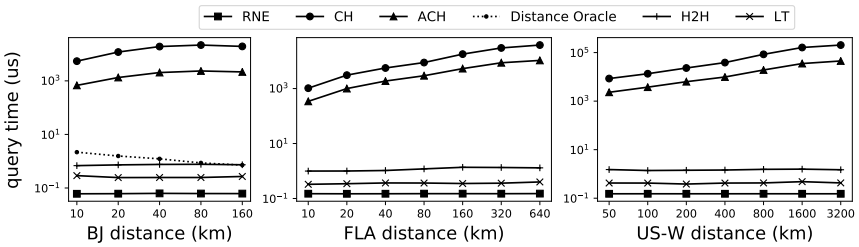


Fig. 13: Query processing time (vary query distance)

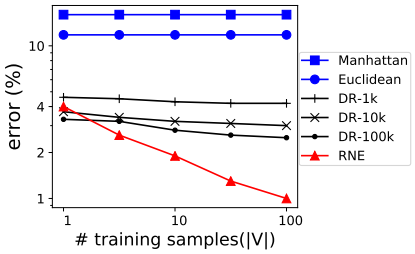


Fig. 14: Comparing with baseline models

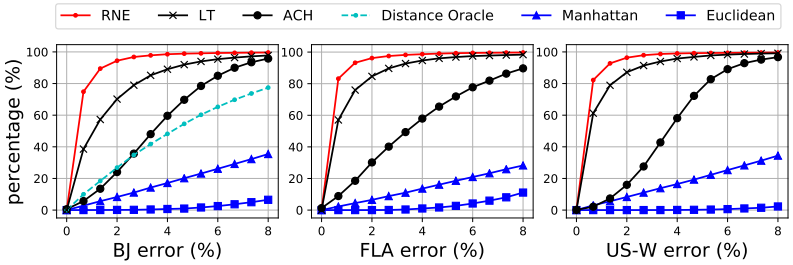


Fig. 15: Error distribution

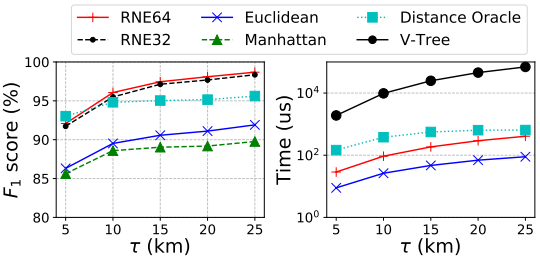


Fig. 16: Range query on **BJ**

(20GB for **US-W**). (4) The index size of RNE is about 1/10 - 1/3 of H2H and takes reasonable training time. Because the memory complexity for RNE is $O(|V| \cdot d)$ where d decides the representation of the model and usually is not large (128 is enough for **US-W**, which takes only 3GB). (5) The index size of LT is 2x of RNE because the space usage is $O(|V| \cdot |U|)$ and $|U|$ is 2x of d for each road network and the index building process is to perform $|U|$ time single source shortest paths search. (6) The index sizes of CH and ACH are the smallest (1GB for **US-W**) but the time costs to build indexes are the highest (more than 10K minutes).

3) Query Error: In this experiment, we evaluate the accuracy (e_{rel} and e_{abs}) of different approximate methods.

Mean relative error. Table III shows the mean value of e_{rel} of every method on 10K randomly chosen queries. We can observe that RNE significantly outperforms Euclidean, Manhattan, Distance Oracle, ACH and LT. Since RNE has much lower query time than the other approximate methods (except 2 baselines) and the best error rate (under 0.7% on all 3 datasets, good enough for most queries), it is most practical in real-world usage.

Percentage error. Figure 15 shows the cumulative percentage of validation samples as the error rate grows. For example for RNE on **BJ**, there are about 93% queries which have a error less than 2% and 99% queries less than 5%. It shows that for datasets of different sizes, the accuracy of RNE significantly outperforms ACH and Distance Oracle and LT, and all of the 4 methods significantly outperform Manhattan and Euclidean.

Vary distance error. Figure 17 shows e_{rel} (line) and e_{abs} (bar) of different distance scales. The setting of the distance scales is the same as that in Section VII-C1. We have the following observations. (1) e_{abs} and e_{rel} of ACH are increasing as the distance increases. That indicates that e_{abs} of ACH grows super-linearly with larger distances. (2) e_{abs} of RNE does not change as distance increases, and thus e_{rel} keeps decreasing. That is because our loss function focuses on the reduction of e_{abs} so the errors for all distance scales are optimized equally. (3) Trend of the error of LT are similar with RNE but the accuracy is worse than RNE. (4) Distance Oracle has the same e_{rel} on various distance scales which are behind ACH. That's because all the block pairs in the oracle satisfy the error bound ϵ. (5) Except for very small distance scales, RNE outperforms ACH. RNE is more accurate than Distance Oracle and LT on every distance scales.

4) Range and kNN Query Performance: We evaluate the performance for range and kNN queries on **BJ**. We compare RNE with two kNN methods Distance Oracle [27] and V-tree [28]. We also compare with two baselines Euclidean and Manhattan with KD-tree as the index. We only show the results of range queries (Figure 16), which are very similar to kNN.

F_1-score. The left part shows the F_1 scores for range queries of various distance threshold τ. We can observe that the accuracy of RNE is slightly higher than Distance Oracle and much higher than Euclidean and Manhattan (with τ varies from $5-25$ km, F_1 of RNE64 and RNE32 are above 90%, $5-10\%$ better than Euclidean and Manhattan). It indicates that RNE gives a high accuracy.

Query time. The right part shows the query time varying τ. We can see that RNE achieves similar performance as Euclidean and Distance Oracle and outperforms V-tree by 10-100 times. Note that the query speeds for Euclidean and Manhattan are extremely fast.

In summary, since the query error keeps reasonable and the query time is very fast for various distance scales, it shows that RNE is more practical.

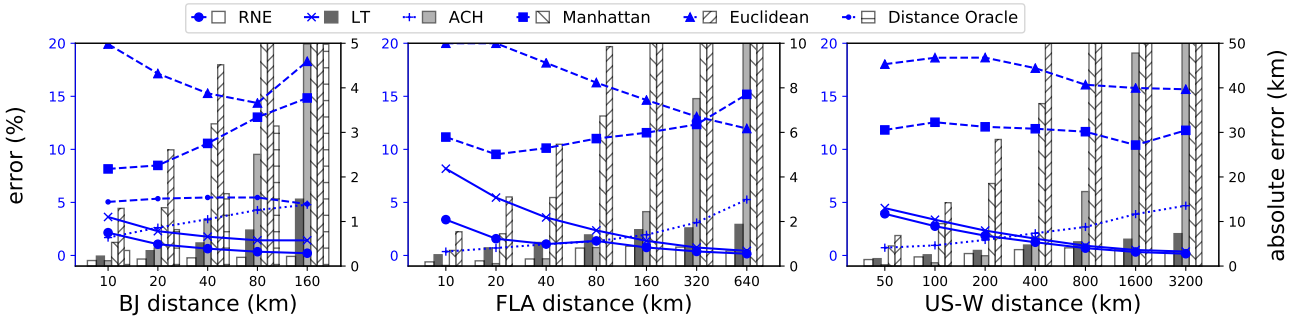


Fig. 17: Mean errors of various distance scales

VIII. Conclusion

In this paper, we propose an embedding based method which embeds every vertex in road networks into a low dimensional space for computing shortest-path distances. We can achieve very low query processing time (in nanoseconds) and very high accuracy (less than 0.7% average error rate) and scale well to large road networks. We leverage a hierarchical learning strategy to significantly speed up the training process and preserve higher-order road network structure. We also propose training sample selection strategies and an active fine-tuning technique to improve the accuracy of the model. The experimental results on real datasets demonstrate that our approach can achieve the fastest querying speed comparing with the state-of-the-arts while achieving a very low error rate.

References

[1] T. Abeywickrama, M. A. Cheema, and D. Taniar. K-nearest neighbors on road networks: a journey in experimentation and in-memory implementation. *VLDB*, 9(6):492–503, 2016.

[2] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *International Symposium on Experimental Algorithms*, pages 230–241. Springer, 2011.

[3] Y. Aumann and Y. Rabani. An o (log k) approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27(1):291–301, 1998.

[4] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route planning in transportation networks. In *Algorithm engineering*, pages 19–80. 2016.

[5] H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast routing in road networks with transit nodes. *Science*, 316(5824):566–566, 2007.

[6] W. Bialek, I. Nemenman, and N. Tishby. Predictability, complexity, and learning. *Neural computation*, 13(11):2409–2463, 2001.

[7] J. Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.

[8] H. Chen, B. Perozzi, Y. Hu, and S. Skiena. Harp: Hierarchical representation learning for networks. In *AAAI*, 2018.

[9] M. M. Deza and M. Laurent. *Geometry of cuts and metrics*, volume 15. 2009.

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

[11] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *International Workshop on Experimental and Efficient Algorithms*, pages 319–333, 2008.

[12] R. Geisberger and D. Schieferdecker. Heuristic contraction hierarchies with approximation guarantee. In *Third Annual Symposium on Combinatorial Search*, 2010.

[13] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *SODA*, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

[14] A. V. Goldberg and R. F. F. Werneck. Computing point-to-point shortest paths from external memory. In *ALENEX/ANALCO*, pages 26–40, 2005.

[15] A. Gupta, I. Newman, Y. Rabinovich, and A. Sinclair. Cuts, trees and l1-embeddings of graphs. *Combinatorica*, 24(2):233–269, 2004.

[16] H. Hu, Y. Zheng, Z. Bao, G. Li, J. Feng, and R. Cheng. Crowdsourced POI labelling: Location-aware result inference and task assignment. In *ICDE*, pages 61–72, 2016.

[17] G. Karypis and V. Kumar. Analysis of multilevel graph partitioning. In *Supercomputing*, pages 29–29. IEEE, 1995.

[18] P. Klein, S. Rao, A. Agrawal, and R. Ravi. An approximate max-flow min-cut relation for undirected multicommodity flow, with applications. *Combinatorica*, 15(2):187–202, 1995.

[19] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)*, 46(6):787–832, 1999.

[20] G. Li, J. Hu, J. Feng, and K. Tan. Effective location identification from microblogs. In *ICDE*, pages 880–891, 2014.

[21] X. Li, Y. Zhao, X. Zhou, and K. Zheng. Consensus-based group task assignment with social impact in spatial crowdsourcing. *Data Sci. Eng.*, 5(4):375–390, 2020.

[22] D. Ouyang, L. Qin, L. Chang, X. Lin, Y. Zhang, and Q. Zhu. When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks. In *SIGMOD*, pages 709–724, 2018.

[23] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *KDD*, pages 701–710. ACM, 2014.

[24] S. Rao. Small distortion and volume preserving embeddings for planar and euclidean metrics. In *Proceedings of the fifteenth annual symposium on Computational geometry*, pages 300–306. ACM, 1999.

[25] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD*, pages 43–54. ACM, 2008.

[26] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest path queries. In *European Symposium on Algorithms*, pages 568–579. Springer, 2005.

[27] J. Sankaranarayanan and H. Samet. Query processing using distance oracles for spatial networks. *IEEE TKDE*, 22(8):1158–1175, 2010.

[28] B. Shen, Y. Zhao, G. Li, W. Zheng, Y. Qin, B. Yuan, and Y. Rao. V-tree: Efficient knn search on moving objects with road-network constraints. In *ICDE*, pages 609–620. IEEE, 2017.

[29] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *WWW*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[30] Y. Wang, G. Li, and N. Tang. Querying shortest paths on time dependent road networks. *VLDB*, 12(11):1249–1261, 2019.

[31] L. Wu, X. Xiao, D. Deng, G. Cong, A. D. Zhu, and S. Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *VLDB*, 5(5):406–417, 2012.

[32] K. Yang, X. Ding, Y. Zhang, L. Chen, B. Zheng, and Y. Gao. Distributed similarity queries in metric spaces. *Data Sci. Eng.*, 4(2):93–108, 2019.

[33] H. Yuan and G. Li. Distributed in-memory trajectory similarity search and join on road network. In *ICDE*, pages 1262–1273, 2019.

[34] H. Yuan, G. Li, Z. Bao, and L. Feng. Effective travel time estimation: When historical trajectories over road networks matter. In *SIGMOD*, pages 2135–2149, 2020.

[35] R. Zhong, G. Li, K.-L. Tan, and L. Zhou. G-tree: An efficient index for knn search on road networks. In *CIKM*, pages 39–48. ACM, 2013.

[36] R. Zhong, G. Li, K.-L. Tan, L. Zhou, and Z. Gong. G-tree: An efficient and scalable index for spatial search on road networks. *IEEE TKDE*, 27(8):2175–2189, 2015.