A Learning Based Approach to Predict Shortest-Path Distances

Jianzhong Qi1, Wei Wang2, Rui Zhang1, Zhuowei Zhao∗

1The University of Melbourne, Melbourne, Australia
2The University of New South Wales, Sydney, Australia

∗{jianzhong.qi@, rui.zhang@, zhuowei1@student.}unimelb.edu.au, ∗weiw@ece.unsw.edu.au

ABSTRACT

Shortest-path distances on road networks have many applications such as finding nearest places of interest (POI) for travel recommendations. To compute a shortest-path distance, traditional approaches traverse the road network to find the shortest path and return the path length. When the distances are needed first (e.g., to rank POIs) while the shortest paths may be computed later (e.g., after a POI is chosen), one may precompute and store the distances, and answer distance queries by simple lookups. This approach, however, falls short in the worst-case space cost – O(n^2) for n vertices even with various optimizations. To address these limitations, we propose to learn an embedding for every vertex that preserves its distances to the other vertices. We then train a multi-layer perceptron (MLP) to predict the distance between two vertices given their embeddings. We thus achieve fast distance predictions without a high space cost. Experimental results on real road networks confirm these advantages. Meanwhile, our approach is up to 97% more accurate than the state-of-the-art approaches for distance predictions.

1 INTRODUCTION

Computing shortest-path distances on road networks with a high efficiency is fundamental for applications such as “finding restaurants within 5 km distance” or “ranking restaurant search results by distance”. Real road networks (e.g., Florida road network [8]) may contain millions of vertices, while thousands of users may issue distance queries at the same time (e.g., Google Maps has over a billion active users [1]). Answering distance queries under such settings poses significant challenges in both space and time costs. We aim to address such challenges in this paper.

Problem formulation. We consider a road network graph \( G = (V, E) \), where \( V \) is a set of \( n \) vertices (road intersections) and \( E \) is a set of \( m \) edges (roads). A vertex \( v_i \in V \) has a pair of geo-coordinates. An edge \( e_{ij} \in E \) connects two vertices \( v_i \) and \( v_j \), and has a weight \( w_{ij} \), which represents the distance to travel across the edge. Fig. 1a shows an example, where \( v_1, v_2, ..., v_5 \) are the vertices, and the numbers on the edges are the weights. For simplicity, in what follows, our discussions assume undirected edges, although our techniques also work for directed edges.

A path \( p_{ij} \) between vertices \( v_i \) and \( v_j \) consists of a sequence of vertices \( v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k \rightarrow v_j \) such that there is an edge between any two adjacent vertices in the sequence. The length of \( p_{ij} \), denoted by \( |p_{ij}| \), is the sum of the weights of the edges between adjacent vertices in \( p_{ij} \), i.e., \( |p_{ij}| = w_{e_{12}} + w_{e_{23}} + \cdots + w_{e_{k-1,k}} \). We are interested in the path of shortest length between \( v_i \) and \( v_j \) with the smallest length, i.e., the shortest path. Its length is the (shortest-path) distance \( d(v_i, v_j) \) between \( v_i \) and \( v_j \), i.e., \( d(v_i, v_j) = |p^*_i, j| \). Consider vertices \( v_1 \) and \( v_5 \) in Fig. 1a. Their distance \( d(v_1, v_5) = 4 \) is the length of path \( v_1 \rightarrow v_4 \rightarrow v_5 \). We aim to predict \( d(v_i, v_j) \) given \( v_i \) and \( v_j \) with a high accuracy and efficiency, which is defined as the shortest-path distance query.

Definition 1.1 (Shortest-path distance query). Given two query vertices \( v_i \) and \( v_j \) in graph \( G \), a shortest-path distance query returns the shortest-path distance between \( v_i \) and \( v_j \), i.e., \( d(v_i, v_j) \).

For simplicity, we use distance to refer to shortest-path distance hereafter as long as the context is clear.

Related work. A simple solution is to use shortest path algorithms (e.g., Dijkstra’s algorithm) to compute the shortest paths and then return the path lengths. In applications such as those mentioned above, the distances are needed first (e.g., to rank restaurants by distance) while the shortest paths may be computed later (e.g., after a restaurant is chosen). Studies (e.g., [3, 7, 12]) thus build data structures to enable fast distance queries without online shortest path computations. Distance labeling is commonly used. Its basic idea is to precompute a vector of (distance) values for each vertex as its distance labeling query without online shortest path computations. Distance labeling is commonly used. Its basic idea is to precompute a vector of (distance) values for each vertex as its distance label. In an extreme case, the distance label of a vertex contains its distances to all other vertices (cf. Fig. 1b). A distance query is answered by a lookup in \( O(1) \) time, but this requires \( O(n^2) \) storage space for the distance labels. Techniques (e.g., 2-hop labeling [7] and highway labeling [12]) are proposed to reduce the label size. However, for general graphs, the worst-case space costs are still \( O(n^2) \) [11].

To avoid the \( O(n^2) \) space cost, approximate techniques are proposed [6, 18, 21], among which landmark labeling [10, 16, 20] is a representative approach. This approach uses a subset of \( k \) (\( k \ll n \)) vertices as the landmarks. Every vertex \( v_i \) stores its distances to these landmarks as its distance label, i.e., a \( k \)-dimensional vector \( (d(v_i, l_1), d(v_i, l_2), \ldots, d(v_i, l_k)) \), where \( l_1, l_2, \ldots, l_k \in L \) represent the landmarks and \( d(\cdot, l) \) represents the distance. At query time, the distance labels of the two query vertices \( v_i \) and \( v_j \) are scanned, where the distances to the same landmark are summed up. The smallest distance sum, i.e., \( \min\{d(v_i, l) + d(v_j, l)\} \),
is returned. In Fig. 1, \(v_2\) and \(v_3\) are chosen as the landmarks (denoted by \(l_1\) and \(l_2\), respectively), and the distance labels are shown in Fig. 1c. The distance between \(v_1\) and \(v_3\) is computed as \(\min(d(v_1, l_1) + d(v_2, l_1), d(v_1, l_2) + d(v_2, l_2)) = \min(3 + 6, 3 + 5) = 8\), which is twice as large as the actual distance between \(v_1\) and \(v_3\) (i.e., 4). Even though landmark labeling reduces the space cost to \(O(kn)\), it may not return the exact distance. How the landmarks are chosen plays a critical role in the distance accuracy. Since finding the \(k\) optimal landmarks is NP-hard [16, 19] such as choosing the vertices that are on more shortest paths as the landmarks. Theoretical results (e.g., [6, 15]) are offered to bound the relationship between the label size and the distance accuracy. On undirected graphs, it is shown [21] that any algorithm with an approximation ratio of \(\alpha < 2c + 1\) \((c \in \mathbb{N}^+)\) must use \(О(n^{1/2})\) space. A structure is proposed [21] using \(O(cn^{1/2})\) space and \(O(cmn^{1/2})\) time to obtain an approximation ratio of \(\alpha = 2c - 1\) and an \(O(c)\) query time. Chechik [6] improves the space cost to \(O(n^{1/2})\) and the query time to \(O(1)\), with an \(O(n^2 + mn^{1/2})\) prepossessing time. These studies are mainly of theoretical interest. They do not offer empirical results.

Our contributions. To preserve more information in the distance labels and obtain higher distance accuracy, we propose to learn an embedding for every vertex as its distance label. Our idea is motivated by recent advances in graph embeddings [4, 5, 9], which show that vertices can be mapped into a latent space where their structural similarity (e.g., the number of common neighboring vertices) can be computed. This motivates us to map the vertices into a latent space to compute their spatial similarity, i.e., shortest-path distances. We make the following contributions:

(i) We propose a learning based model \(vdist2vec\) to predict vertex distances. This model learns vertex embeddings while jointly trains a multi-layer perceptron (MLP) to predict vertex distances. It has an \(O(k)\) distance prediction time and an \(O(n)\) storage space, where \(k\) is a small constant denoting the vertex embedding dimensionality. Our model is highly accurate, since the embeddings are guided by distance predictions directly.

(ii) We further propose two models \(vdist2vec-L\) and \(vdist2vec-S\) with an improved loss function and an improved model structure to optimize the embeddings for different types of vertices.

(iii) We perform experiments on real road networks. The results show that, comparing with state-of-the-art approaches, our models reduce the distance prediction errors by up to 97%.

2 PROPOSED MODEL

\(vdist2vec\). As Fig. 2 shows, our \(vdist2vec\) model takes two vertices \(v_i\) and \(v_j\) as the input, which are represented as two \(|V|\)-one-hot vectors \(h_i\) and \(h_j\). The next layer is an embedding layer for representation learning. This layer has \(k\) nodes, and its weight matrix is a \(|V| \times k\) matrix. To be used as the vertex vectors for all vertices, denoted by \(V = \{v_1^T, v_2^T, \ldots, v_{|V|}^T\}^T\). Multiplying \(h_i\) by \(V\) yields \(v_i\) (\(v_j\)):

\[
v_i = h_i V
\]

Vectors \(v_i\) and \(v_j\) are then fed into a distance prediction network (i.e., an MLP) to predict the distance between \(v_i\) and \(v_j\). The loss function \(L_d\) is the mean square error on the actual vertex distances \(d(v_i, v_j)\) and the predicted distances \(\hat{d}_{i,j}\):

\[
L_d = \mathbb{E}_P[f(d(v_i, v_j) - \hat{d}_{i,j})^2]
\]

At training, the vertex representation matrix \(V\) is randomly initialized. Vertex pairs are fed into the network in batches to train the MLP. The training loss \(L_d\) will be propagated back to optimize the MLP and the vertex representations in \(V\).

At query time, the query vertex vectors \(v_i\) and \(v_j\) are fetched from \(V\) and fed into the MLP to make a distance prediction.

(a) Error distribution (DG)  (b) \(vdist2vec-S\)

Figure 3: Prediction error distribution and \(vdist2vec-S\)

Next, we optimize the prediction accuracy further. Our motivation comes from an observation on the distance prediction error distributions. In Fig. 3a, we plot the normalized absolute distance prediction errors of \(vdist2vec\) on a real dataset (DG, cf. Section 3). The x-axis represents vertex pairs (sorted by the prediction errors) and the y-axis represents their corresponding prediction errors. We see that a very small portion (e.g., less than 1%) of the vertex pairs have much larger prediction errors (see the spike to the right of the figure) than the other vertex pairs. Such vertex pairs dominate the training error and force the model to focus on them. To optimize the overall prediction accuracy, we need to guide the model to attend more to the other vertex pairs.

\(vdist2vec\) - L. Our first optimization is a novel loss function denoted by \(L_n\) to shrink the larger errors:

\[
L_n = \mathbb{E}_P[f_S(d(v_i, v_j) - \hat{d}_{i,j})]
\]

Function \(f_S(\cdot)\) is motivated by the Huber loss and is continuously differentiable at \(|x| = \delta\). We set \(\delta\) as the top 1% largest prediction error after each epoch. If \(|x| > \delta\), \(\frac{1}{2}(x^2 + \delta^2) \leq x^2\) shrinks the error. Replacing \(L_d\) with \(L_n\) results in \(vdist2vec\)-L.

\(vdist2vec\) - S. Our second optimization is motivated by ensemble learning. As Fig. 3b shows, we replace the last hidden layer of \(vdist2vec\) with four separate MLPs, each focusing on producing distance predictions in the ranges of \((0, 100)\), \((0, 900)\), \((0, 9000)\), and \((0, d_{max} = 10000)\), where \(d_{max}\) is the road network diameter. This is done by multiplying (\(\cdot\)) the sigmoid output of each MLP with 100, 900, 9000, and \(d_{max} = 10000\), respectively. The output of the MLPs are summed up to produce the final predictions. We name this model \(vdist2vec\)-S. Its advantage is that each MLP can focus on vertex pairs in different distance ranges, making it easier to learn more accurate predictions.

As shown in Fig. 3a, the error distributions of \(vdist2vec\)-L and \(vdist2vec\)-S are less skewed than that of \(vdist2vec\).
Handling large road networks. Our models compute a \( |V| \times k \) embedding matrix. This is cheaper than a \( |V|^2 \) matrix for all vertex distances. However, we still need to train over \( |V|^2 \) pairs of vertices. Next, we reduce the number of training pairs.

We cluster (e.g., using k-means) the vertices into \( |V_c| \) clusters based on their geo-coordinates, where \( |V_c| \) is a small constant. The vertex nearest to each cluster center is chosen as a center vertex. We train our models over the center vertices. Given query vertices \( v_i \) and \( v_j \), their distance \( d_{ij} \) is approximated by their distances to their cluster center vertices \( v_{ic} \) and \( v_{jc} \) (which are precomputed) plus the predicted distance \( \hat{d}_{ic,jc} \) of \( v_{ic} \) and \( v_{jc} \):

\[
\hat{d}_{ij} = \lambda_1 \cdot d(v_i, v_{ic}) + \hat{d}_{ic,jc} + \lambda_2 \cdot d(v_{jc}, v_j)
\]

There are two coefficients \( \lambda_1 \) and \( \lambda_2 \) to weight the contributions of \( d(v_i, v_{ic}) \) and \( d(v_{jc}, v_j) \) based on the relative positions of the vertices (cf. Fig. 4a). To learn \( \lambda_1 \) and \( \lambda_2 \), we build another neural network as shown in Fig. 4b, where \( \hat{d}_{ic,jc} \) and \( \hat{d}_{ic,jc} \) return the difference in latitude and longitude between two vertices, respectively. This neural network feeds the coordinate difference between \( v_i \) and \( v_{ic} \) and the coordinate difference between \( v_j \) and \( v_{jc} \) into two MLPs to predict \( \lambda_1 \) and \( \lambda_2 \), respectively. The last layer of each MLP uses a \( \tanh \) activation function, which maps \( \lambda_1 \) and \( \lambda_2 \) into the range of \((-1, 1)\). The output of these two MLPs is multiplied \((\ast)\) with \( d(v_i, v_{ic}) \) and \( d(v_{jc}, v_j) \), and the products are added \((\ast)\) with \( \hat{d}_{ic,jc} \) to produce \( \hat{d}_{ij} \), which implements Equation 4. For training, we use loss function \( L_d \) but with only a sampled subset of non-center vertices (e.g., \( |V_c||V| \) pairs), since the input space (i.e., coordinate difference) is now much smaller.

![Figure 4: Distance prediction for large road networks](image)

Handling updates. Our models can be rebuilt in 13 hours for road networks with over a million vertices (Section 3). This allows us to handle a low update frequency by periodic rebuilds. Our models can also provide distance predictions upon vertex or edge updates without rebuilding, although the accuracy may drop. We leave more robust update handling for future study.

Cost analysis. We consider an MLP to have an \( O(1) \) space cost for its parameters and an \( O(1) \) time cost for inference. Such costs depend mainly on the model size rather than the input size. Also, the inference can be done by GPUs efficiently. Then, our models can be trained in \( O(|V|^2) \) time \((O(|V_c||V|)\) for large road networks). Our models take \( O(k|V|) \) space for the embeddings. They take \( O(k) \) time to read and feed the query vertex embeddings into the MLP for distance prediction in \( O(1) \) time.

3 EXPERIMENTS

We run experiments on a Linux PC with an Intel(R) Xeon(R) E5-2630 V3 CPU (2.40GHZ), a GeForce GTX TITAN X GPU, and 32GB memory. All models are implemented with Python 2.7.12. The neural networks are implemented with Tensorflow 1.13.1.

Datasets. We use six road network datasets as summarized in Table 1, where \( dgr \) denotes the average degree and \( d_{max} \) denotes the diameter. All datasets are undirected except for MB.

### Table 1: Datasets

| Dataset                    | \( |V| \) | \( |E| \) | \( dgr \) | \( d_{max} \) |
|----------------------------|--------|--------|--------|----------|
| Dongguan, China (DG) [14]  | 38K    | 11K    | 2.76   | 96km     |
| Florida, USA (FL) [8]      | 1.07M  | 1.35M  | 2.36   | 1,200km  |
| Melbourne, Australia (MB) [2] | 3.6K | 4.1K    | 1.14   | 6km      |
| New York City, USA (NY) [8] | 264K   | 366K   | 2.80   | 160km    |
| Shanghai, China (SH) [14]  | 74K    | 100K   | 2.70   | 127km    |
| Surat, India (SU) [14]     | 2.5K   | 3.6K   | 2.88   | 50km     |

Baselines. We compare with five baselines: **landmark-bt** [19]; it uses the top-k vertices passed by the largest numbers of shortest paths between the vertex pairs as the landmarks. **landmark-km**; it uses the k vertices that are the closest to the vertex k-means centroids (computed in Euclidean space) as the landmarks; **ado** [18]; it recursively partitions the vertices into subsets of well separated vertices and stores the distance between subsets to approximate the distance between vertices (we tune its approximation parameter \( \epsilon \) such that it has a similar space cost to ours); **geodnn** [13]; it trains an MLP to predict the distance of two vertices given their geo-coordinates (we use its recommended settings); **node2vec** [17]; it uses node2vec [9] to learn vertex embeddings and trains an MLP to predict vertex distances given the learned embeddings (we use its recommended settings).

Hyperparameters. For our models, the MLP distance prediction component has two hidden layers of 100 and 20 nodes, respectively. We use ReLU as the activation function for the hidden layers and sigmoid for the output layer. We set the batch size to be \( |V| \) (we find that a large batch size helps the training efficiency without impacting the prediction accuracy). We initialize the MLP parameters using the truncated normal distribution with \( 0 \) as the mean and 0.01 as the standard deviation. The training data is randomly shuffled. We train our model in 20 epochs with early stopping using AdamOptimizer and a learning rate of 0.01.

Each ensemble MLP of vdist2vec-S has a layer of 20 nodes.

In all approaches except node2vec, we use \( k = 2\%|V| \) for DG, MB, and SU, \( k = 0.05\%|V| \) for SH, and \( k = 0.005\%|V| \) for FL and NY. For node2vec, we use \( k = 128 \) as suggested by [17].

Evaluation metrics. We predict the distance between every two vertices in each dataset and measure the **mean absolute error** (MAE, in meters), **mean relative error** (MRE), **precomputation/training time** (PT), and **average distance prediction (query) time** (QT). The ground truth distances are precomputed using the contraction hierarchy algorithm.

Overall results. Table 2 shows the prediction errors. Our models outperform the baseline models across all six datasets and reduce the MAE and MRE by up to 97% and 99% (5 vs. 192 and 0.006 vs. 0.488 for vdist2vec-S and landmark-bt on MB). On NY, landmark-km has a slightly lower MAE than ours, while our MRE is still lower (by more than 50%).

The advantage of our models comes from their capability to learn the vertex distances and preserve them in the embeddings. Landmark-bt and landmark-km rely on the landmarks and may not preserve the distance for all vertices. ADO is designed to control the relative error. It yields the lowest MRE among the baselines on most datasets, but its MAE may be large. Geodnn uses Euclidean distance to approximate shortest-path distance. It suffers on large road networks (e.g., FL and NY) with rivers and large detours. Node2vec focuses on the neighborhood of the vertices. It also suffers on large road networks such as NY (it cannot train on FL in 48 hours which is marked as "QT").

In Table 2, we also show the space required to store the learned embeddings and model parameters on the FL dataset. Geodnn has the smallest space requirement, as it only stores the MLP.
We proposed a representation learning based approach for the shortest-path distance problem. Our approach learns vertex embeddings that preserve the distances between vertices, which are fed into an MLP to predict the distance, which takes a constant time. Experimental results show that our approach is highly efficient. It reduces the distance prediction errors by up to 97% comparing with the state-of-the-art. For future work, we plan to extend our techniques to more types of (and larger) graphs such as social networks. We also plan to study real-time updates for learning based distance prediction models.

ACKNOWLEDGMENTS

This work is partially supported by Australian Research Council (ARC) Discovery Projects DP180102050 and DP180103332.

REFERENCES


