Short paper

A Learning Based Approach to Predict Shortest-Path Distances

Jianzhong Qi[1], Wei Wang[2], Rui Zhang[1], Zhuowei Zhao[1*]
[1]The University of Melbourne, Melbourne, Australia
[2]The University of New South Wales, Sydney, Australia
[1]{jianzhong.qi@, rui.zhang@, zhuoweiz1@student.}unimelb.edu.au, [2]weiw@cse.unsw.edu.au

ABSTRACT

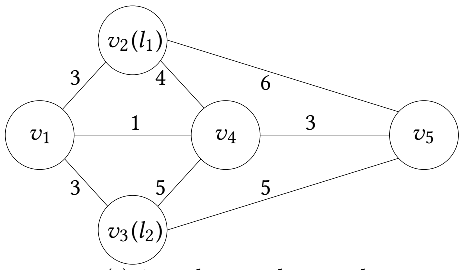
Shortest-path distances on road networks have many applications such as finding nearest *places of interest* (POI) for travel recommendations. To compute a shortest-path distance, traditional approaches traverse the road network to find the shortest path and return the path length. When the distances 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-cast 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 $G = \langle V, E\rangle$, 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 $e_{i,j} \in E$ connects two vertices v_i and v_j, and has a *weight* $e_{i,j}.w$, 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 $p_{i,j}$ between vertices v_i and v_j consists of a sequence of vertices $v_i \to v_1 \to v_2 \to \ldots \to v_x \to v_j$ such that there is an edge between any two adjacent vertices in the sequence. The *length* of $p_{i,j}$, denoted by $|p_{i,j}|$, is the sum of the weights of the edges between adjacent vertices in $p_{i,j}$, i.e., $|p_{i,j}| = e_{i,1}.w + e_{1,2}.w + \ldots + e_{x,j}.w$. We are interested in the path $p^*_{i,j}$ 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



(a) A road network example

	v_1	v_2	v_3	v_4	v_5
v_1	0	3	3	1	4
v_2	3	0	6	4	6
v_3	3	6	0	4	5
v_4	1	4	4	0	3
v_5	4	6	5	3	0

(b) Distance labeling

	$v_2(l_1)$	$v_3(l_2)$
v_1	3	3
v_2	0	6
v_3	6	0
v_4	4	4
v_5	6	5

(c) Landmark labeling

Figure 1: Shortest-path distance problem

distance $d(v_1, v_5) = 4$ is the length of path $v_1 \to v_4 \to v_5$. 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 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 $\langle d(v_i, l_1), d(v_i, l_2), \ldots, d(v_i, l_k)\rangle$, where $l_1, l_2, \ldots, l_k \in L$ represent the landmarks and $d(\cdot)$ 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., $\min\{d(v_i, l) + d(v_j, l) | l \in L\}$,

[*]The authors are ordered alphabetically.

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_5 is computed as $\min\{d(v_1, l_1) + d(v_5, l_1), d(v_1, l_2) + d(v_5, l_2)\} = \min\{3+6, 3+5\} = 8$, which is twice as large as the actual distance between v_1 and v_5 (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], heuristics are proposed [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 $\Omega(n^{1+\frac{1}{c}})$ space. A structure is proposed [21] using $O(cn^{1+\frac{1}{c}})$ space and $O(cmn^{\frac{1}{c}})$ time to obtain an approximation ratio of $\alpha = 2c - 1$ and an $O(c)$ query time. Chechik [6] improves the space cost to $O(n^{1+\frac{1}{c}})$ and the query time to $O(1)$, with an $O(n^2 + mn^{\frac{1}{2}})$ 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 $O(kn)$ 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

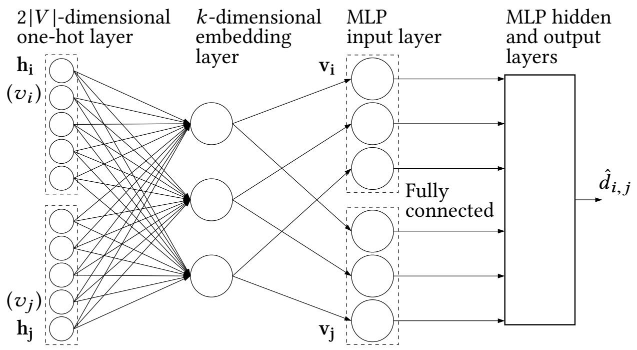


Figure 2: Vdist2vec model structure

Vdist2vec. As Fig. 2 shows, our vdist2vec model takes two vertices v_i and v_j as the input, which are represented as two size-$|V|$ *one-hot vectors* $\mathbf{h_i}$ and $\mathbf{h_j}$. The next layer is an *embedding layer* for representation learning. This layer has k nodes, and its weight matrix is a $|V| \times k$ ($2|V| \times k$ for directed graphs) matrix to be used as the vertex vectors for all vertices, denoted by $\mathbf{V} = [\mathbf{v_1}^T, \mathbf{v_2}^T, ..., \mathbf{v_{|V|}}^T]^T$. Multiplying $\mathbf{h_i}$ ($\mathbf{h_j}$) by $\mathbf{V}$ yields $\mathbf{v_i}$ ($\mathbf{v_j}$):

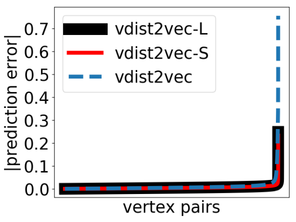
$$\mathbf{v_i} = \mathbf{h_i}\mathbf{V} \tag{1}$$

Vectors $\mathbf{v_i}$ and $\mathbf{v_j}$ 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 $\mathcal{L}_d$ is the mean square error on the actual vertex distances $d(v_i, v_j)$ and the predicted distances $\hat{d}_{i,j}$:

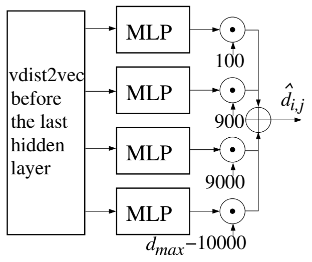
$$\mathcal{L}_d = E_{\mathcal{P}}\left[(d(v_i, v_j) - \hat{d}_{i,j})^2\right] \tag{2}$$

At training, the vertex representation matrix $\mathbf{V}$ is randomly initialized. Vertex pairs are fed into the network in batches to train the MLP. The training loss $\mathcal{L}_d$ will be propagated back to optimize the MLP and the vertex representations in $\mathbf{V}$.

At query time, the query vertex vectors $\mathbf{v_i}$ and $\mathbf{v_j}$ are fetched from $\mathbf{V}$ 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 $\mathcal{L}_n$ to shrink the larger errors:

$$\mathcal{L}_n = E_{\mathcal{P}}\left[f_\delta(d(v_i, v_j) - \hat{d}_{i,j})\right] \tag{3}$$

$$f_\delta(x) = \begin{cases} \delta|x|, \text{if } |x| \le \delta \\ \frac{1}{2}(x^2 + \delta^2), \text{otherwise} \end{cases}$$

Function $f_\delta(\cdot)$ is motivated by the *Huber loss* and is continuously differentiable at $x = \delta$. We set δ as the top 1% largest prediction error after each epoch. If $|x| > \delta$, $\frac{1}{2}(x^2 + \delta^2) \le x^2$ which shrinks the error. Replacing $\mathcal{L}_d$ with $\mathcal{L}_n$ 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 ("$\odot$") 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 in that each MLP can focus on vertex pairs in different distance ranges, making it easier to learn more accurate predictions.

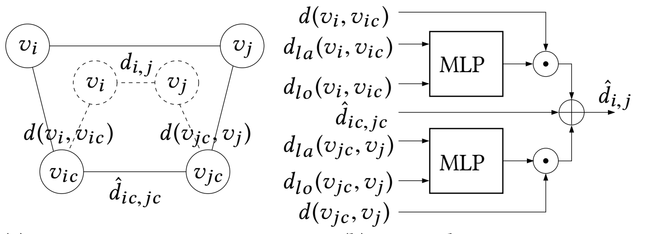
As shown in Fig. 3a, the error distributions of vdist2vec-L and vdist2vec-S are less skewed than that of vdist2vec.

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 $\tilde{d}_{i,j}$ 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}:

$$\tilde{d}_{i,j} = \lambda_1 \cdot d(v_i, v_{ic}) + \hat{d}_{ic,jc} + \lambda_2 \cdot d(v_{jc}, v_j) \tag{4}$$

There are two coefficients λ_1 and λ_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 λ_1 and λ_2, we build another neural network as shown in Fig. 4b, where $d_{la}(\cdot)$ and $d_{lo}(\cdot)$ 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 λ_1 and λ_2, respectively. The last layer of each MLP uses a *tanh* activation function, which maps λ_1 and λ_2 into the range of $(-1, 1)$. The output of these two MLPs is multiplied ("$\odot$") with $d(v_i, v_{ic})$ and $d(v_{jc}, v_j)$, and the products are added ("$\bigoplus$") with $\hat{d}_{ic,jc}$ to produce $\tilde{d}_{i,j}$, which implements Equation 4. For training, we use loss function $\mathcal{L}_d$ 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.



(a) Distance computation (b) Network structure

Figure 4: Distance prediction for large road networks

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 $\overline{dgr}$ denotes the average degree and d_{max} denotes the diameter. All datasets are undirected except for MB.

Table 1: Datasets

Dataset	$\lvert V \rvert$	$\lvert E \rvert$	$\overline{dgr}$	d_{max}
Dongguan, China (**DG**) [14]	8K	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 ϵ 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 "OT").

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

Table 2: Mean Absolute and Relative Errors

		DG		MB		SU		FL			NY		SH	
		MAE	MRE	MAE	MRE	MAE	MRE	MAE	MRE	Size	MAE	MRE	MAE	MRE
baseline	landmark-bt	2,234	0.442	192	0.488	468	0.281	OT	OT	OT	24,851	0.167	6,144	0.554
	landmark-km	74	0.028	15	0.040	142	0.090	58,869	0.113	428 MB	**13,431**	0.105	1,314	0.159
	ado	2,108	0.057	75	0.072	642	0.074	175,571	0.052	431 MB	31,737	0.064	4,539	0.147
	geodnn	1,566	0.092	95	0.097	442	0.108	363,661	0.317	**17 MB**	207,694	0.862	14,842	0.990
	node2vec	2,329	0.199	118	0.161	658	0.175	OT	OT	OT	217,400	0.703	19,465	1.276
proposed	vdist2vec	135	0.015	12	0.014	83	0.027	34,757	0.027	30 MB	16,805	**0.052**	1,290	**0.068**
	vdist2vec-L	75	0.015	6	0.014	50	0.024	34,860	0.027	30 MB	16,793	**0.052**	1,290	**0.068**
	vdist2vec-S	**71**	**0.011**	**5**	**0.006**	**49**	**0.014**	**34,329**	**0.026**	33 MB	16,649	**0.052**	**1,287**	**0.068**

Table 3: Preprocessing and Query Times

		DG		MB		SU		FL		NY		SH	
		PT	QT	PT	QT	PT	QT	PT	QT	PT	QT	PT	QT
baseline	landmark-bt	0.1h	5.832μs	62.7s	4.579μs	32.6s	4.463μs	OT	OT	39.6h	11.712μs	14.7h	8.423μs
	landmark-km	**2.2s**	6.024μs	**0.7s**	4.439μs	**0.4s**	4.322μs	**145.1s**	63.718μs	**66.3s**	15.343μs	**9.5s**	8.930μs
	ado	1.0h	1.080ms	0.2h	0.490ms	195.7s	0.356ms	1.5h	0.110ms	0.8h	0.148ms	138s	0.053ms
	geodnn	0.9h	**0.366μs**	0.2h	**0.396μs**	0.2h	**0.375μs**	0.1h	**0.444μs**	0.1h	**0.458μs**	0.1h	**0.432μs**
	node2vec	2.2h	0.829μs	0.9h	0.820μs	0.5h	0.809μs	OT	OT	26.3h	0.751μs	2.8h	0.781μs
proposed	vdist2vec	2.3h	1.039μs	0.9h	0.644μs	0.4h	0.589μs	12.5h	3.981μs	6.1h	1.215μs	0.2h	0.797μs
	vdist2vec-L	2.3h	1.039μs	0.9h	0.644μs	0.5h	0.589μs	12.5h	3.981μs	6.1h	1.215μs	0.2h	0.797μs
	vdist2vec-S	2.8h	1.366μs	1.1h	1.005μs	0.6h	0.921 μs	12.5h	3.981μs	6.1h	1.215μs	0.2h	0.797μs

parameters. Our models store MLP parameters and center vertex embeddings, which require more space than geodnn but less than landmark-km and ado. Note that, if our models learn vertex embeddings for a full graph, we expect a slightly higher space requirement than landmark-km and ado.

Table 3 shows the preprocesing (training) time PT and distance prediction (query) time QT. In terms of PT, the landmark approaches are much faster on the small road networks DG, MB, and SU. Their precomputation is simpler than the training of the learning based models. On large road networks FL, NY, and SH, our models use the proposed clustering based strategy ($|V_c| = 0.4\%|V|$ and 100,000 random vertex pairs to learn λ_1 and λ_2), which reduces the training time significantly. Ado and node2vec need to run on the full road networks. Their PT grows with the road network size. For geodnn, we randomly sample 100,000 vertex pairs for training on the large road networks. It does not learn vertex embeddings and hence has a lower PT.

In terms of QT, the learning based approaches are highly efficient because their distance prediction is a simple forward propagation, which can be done by GPUs efficiently. Geodnn is the fastest, as its input layer only has four nodes (i.e., two geo-coordinates). The other learning based approaches including ours have very similar MLP structures and input sizes which are larger than that of geodnn. Thus, their QT are similar and are larger than that of geodnn. Ado has the largest QT because it needs to first locate the subsets containing the query vertices.

Among our models, vdist2vec-S yields the smallest distance prediction errors, as it can cope with distances in varying ranges. This advantage comes with a larger PT. In contrast, vdist2vec-L has almost the same PT as vdist2vec but achieves smaller distance prediction errors due to its optimized loss function.

4 CONCLUSIONS

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 only take an $O(kn)$ storage space. At query time, the vertex embeddings 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

[1] 2017. Google announces over 2 billion monthly active devices on Android. https://www.theverge.com/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users.

[2] 2017. Planet OSM. https://planet.osm.org.

[3] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*.

[4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning graph representations with global structural information. In *CIKM*.

[5] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *AAAI*.

[6] Shiri Chechik. 2015. Approximate distance oracles with improved bounds. In *STOC*.

[7] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2003. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.* 32, 5 (2003), 1338–1355.

[8] Camil Demetrescu, Andrew Goldberg, and David Johnson. 2006. 9th DIMACS implementation challenge–Shortest Paths. *American Math. Society* (2006).

[9] Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable feature learning for networks. In *KDD*.

[10] Andrey Gubichev, Srikanta Bedathur, Stephan Seufert, and Gerhard Weikum. 2010. Fast and accurate estimation of shortest paths in large graphs. In *CIKM*.

[11] Minhao Jiang, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, and Yanyan Xu. 2014. Hop doubling label indexing for point-to-point distance querying on scale-free networks. *PVLDB* 7, 12 (2014), 1203–1214.

[12] Ruoming Jin, Ning Ruan, Yang Xiang, and Victor Lee. 2012. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *SIGMOD*.

[13] Ishan Jindal, Xuewen Chen, Matthew Nokleby, Jieping Ye, et al. 2017. A unified neural network approach for estimating travel time and distance for a taxi trip. *arXiv preprint arXiv:1710.04350* (2017).

[14] Alireza Karduni, Amirhassan Kermanshah, and Sybil Derrible. 2016. A protocol to convert spatial polyline data to network formats and applications to world urban road networks. *Scientific Data* 3 (2016), 160046.

[15] David Peleg. 2000. Proximity-preserving labeling schemes. *Journal of Graph Theory* 33, 3 (2000), 167–176.

[16] Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. 2009. Fast shortest path distance estimation in large networks. In *CIKM*.

[17] Fatemeh Salehi Rizi, Joerg Schloetterer, and Michael Granitzer. 2018. Shortest path distance approximation using deep learning techniques. In *ASONAM*.

[18] Jagan Sankaranarayanan and Hanan Samet. 2009. Distance oracles for spatial networks. In *ICDE*.

[19] Frank W. Takes and Walter A. Kosters. 2014. Adaptive landmark selection strategies for fast shortest path computation in large real-world graphs. In *WI-IAT*.

[20] Liying Tang and Mark Crovella. 2003. Virtual landmarks for the internet. In *SIGCOMM*.

[21] Mikkel Thorup and Uri Zwick. 2005. Approximate distance oracles. *J. ACM* 52, 1 (2005), 1–24.