Spatial Structure-Aware Road Network Embedding via Graph Contrastive Learning

Yanchuan Chang
University of Melbourne
Australia
yanchuanc@student.unimelb.edu.au

Egemen Tanin
University of Melbourne
Australia
etanin@unimelb.edu.au

Xin Cao
University of New South Wales
Australia
xin.cao@unsw.edu.au

Jianzhong Qi
University of Melbourne
Australia
jianzhong.qi@unimelb.edu.au

ABSTRACT

Road networks are widely used as a fundamental structure in urban transportation studies. In recent years, with more research leveraging deep learning to solve conventional transportation problems, how to obtain robust road network representations (i.e., embeddings) applicable for a wide range of applications became a fundamental need. Existing studies mainly adopt graph embedding methods. Such methods, however, foremost learn the topological correlations of road networks but ignore the spatial structure (i.e., spatial correlations) which are also important in applications such as querying similar trajectories. Besides, most studies learn task-specific embeddings in a supervised manner such that the embeddings are sub-optimal when being used for new tasks. It is inefficient to store or learn dedicated embeddings for every different task in a large transportation system.

To tackle these issues, we propose a model named *SARN* to learn generic and task-agnostic road network embeddings based on self-supervised contrastive learning. We present (i) a spatial similarity matrix to help learn the spatial correlations of the roads, (ii) a sampling strategy based on the spatial structure of a road network to form self-supervised training samples, and (iii) a two-level loss function to guide SARN to learn embeddings based on both local and global contrasts of similar and dissimilar road segments. Experimental results on three downstream tasks over real-world road networks show that SARN outperforms state-of-the-art self-supervised models consistently and achieves comparable (or even better) performance to supervised models.

1 INTRODUCTION

Road networks are a frequently used structure in a wide range of urban applications, such as querying shortest-path distances [16, 28] and similar trajectories [21, 41]. They have been commonly represented as graphs, where graph algorithms such as *Dijkstra's algorithm* are used for the queries. In road network graphs, each individual vertex or edge carries limited information. As such, graph algorithms often require a traversal over many vertices and edges to answer a query, which at times lacks efficiency.

Motivated by recent advances in graph representation learning [12, 13, 31], we learn embeddings to encode more information into the vector representation of each road segment (modeled as a graph vertex). A road segment embedding is a vectorized representation of the road segment, i.e., we compute a multidimensional real-valued vector to represent a road segment. The embeddings have a continuous value domain and a possibly high dimensionality (e.g., 128 dimensions). They offer a stronger capacity to encode more information (e.g., spatial properties and local topology) of the road segments, compared with a tuple of several values (e.g., road ID and type). This property enables more efficient and accurate queries on road networks, which is attractive to many applications (e.g., finding users with similar commute trajectories for carpooling). For example, computing trajectory similarity is a fundamental operation which plays a critical role in evaluating the query predicate of trajectory queries [7, 36, 43]. We can reduce the time complexity of trajectory similarity computation to linear time by directly comparing the embeddings (e.g., with L1-norm), while traditional methods [5, 18] may need a quadratic time since they compute the distance between every pair of points on two trajectories.

Most existing road network embeddings [17, 33, 40] are learned with given downstream tasks. For example, Wu et al. [40] learn road segment embeddings for different downstream tasks such as road type prediction. Jepsen et al. [17] learn road segment embeddings for driving speed prediction. Such embeddings are learned in an end-to-end fashion with task-specific supervision signals. They may become inapplicable (or sub-optimal) when a new downstream task arises.

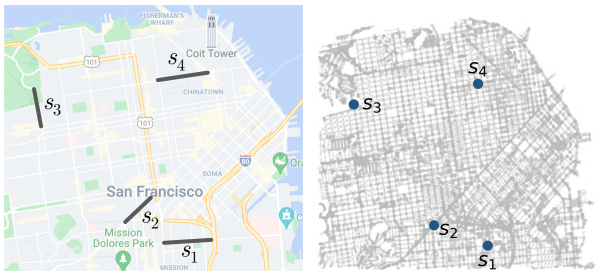
Another stream of studies [35, 44] learn embeddings independently from downstream tasks. They avoid task-specific supervision signals by: (i) using self-supervised graph embedding techniques (e.g., *node2vec* [12]) which focus on the graph topology but ignore the spatial structure of the road networks [44], or (ii) using road segment distances and types (e.g., motorway) as the supervision signals, which ignore the graph topology [35].

In this study, we present a self-supervised learning approach to embed road networks without requiring task-specific supervision signals. We aim to achieve *generic road segment embeddings that preserve both the graph topology and spatial structure information*, which can be used in a variety of applications, such as shortest-path distance query and trajectory similarity computation.

We propose a *spatial structure-aware road network embedding* (SARN) model based on *graph contrastive learning* (GCL), motivated by the strong results of GCL models [14, 29, 42, 49]. The basic idea of GCL is that, given a graph G, we generate a pair of graph variants (i.e., *graph views*) by augmenting G, e.g., randomly masking some edges or vertex attributes. Then, a graph encoder $\mathcal{F}$ is applied on each view to map the vertices (road segments in our problem) to the embeddings. The mapping aims to generate similar embeddings for the same vertex $s_i \in G$ in both graph views and dissimilar embeddings for the different vertices. This learns embeddings in a task-agnostic manner.

However, there are no GCL models designed for road networks and it is sub-optimal to directly apply existing GCL models (e.g., GraphCL [42] and GCA [49]) on road networks due to the following reasons.

- Existing GCL models are designed for general graphs which do not consider the spatial structure of road networks. They randomly sample vertex (i.e., road segment) pairs and give an



(a) Map of San Francisco **(b) Road network graph**

Figure 1: A road network example

equal focus to differentiating the embeddings of the vertices in each pair. During this process, the spatial structure of road networks is ignored. Consider four road segments s_1, s_2, s_3, and s_4 in Fig. 1a, with the corresponding graph shown in Fig. 1b. When learning the embedding of s_1, existing GCL models consider that it is equally important to differentiate s_1 from s_2, s_3, and s_4. However, since s_2 is much closer to s_1 than s_3 and s_4 in spatial distance, we should treat s_2 differently and allow it to have a more similar embedding to that of s_1.

- Existing GCL models learn the embedding of a vertex by aggregating the embeddings of its topologically connected vertices. This also ignores the spatial correlations between the road segments when applying GCL models on road network graphs. For example, even though s_1 and s_2 are close in spatial distance and they share similar directions, they are many hops away in the road network graph in Fig. 1b. As a result, existing GCL models may not learn similar embeddings for the two road segments, while their spatial similarity is important in problems such as querying similar trajectories, which need to be captured and encoded into the embeddings.

To introduce spatial information into GCL, we design novel components for each key GCL step in our SARN model as follows:

The *spatial similarity adjacency matrix* records the similarity between road segments based on their spatial properties, e.g., locations and angles. We incorporate this matrix into the graph encoder in SARN to guide the message passing process and learn embeddings that reflect the similarities in the spatial properties.

The *spatial importance-based graph augmentation strategy* helps generate graph views by masking the less important edges and retaining edges between road segments of higher importance (e.g., motorways that connect many parts of the road network) and sharing more similar spatial properties.

The *spatial distance-based negative sampling strategy* generates training vertex pairs with a focus on the road segments close to a target segment s_i for embedding learning. We partition the road network space with a grid and fetch multiple recently learned embeddings in the grid cell of s_i as our *local negative samples*, and we generate an aggregated embedding for each of the other cells as a *global negative sample*.

The *two-level contrastive loss function* takes both local and global contrasts into consideration by leveraging the two types of negative samples. The local contrastive loss focuses on distinguishing the embedding of s_i from those of the local negative samples, while the global contrastive loss focuses on distinguishing the embedding of s_i from those of the global negative samples. The two loss terms together produce embeddings that are more similar (but still distinguishable) for the road segments in the same local area than those for the road segments in different areas, thus better preserving the spatial structures.

To summarize, we make the following contributions:

(1) We propose a model named SARN which is the first road network embedding model based on graph contrastive learning. The contrastive learning-based process enables embedding learning without manually labelled data, while the learned embeddings can be readily used in a wide range of downstream tasks without fine-tuning.
(2) To encode spatial information, we design four novel components for each key learning step of SARN including a spatial similarity-based graph, a spatial importance-based graph augmentation strategy, a spatial distance-based negative sampling strategy and a two-level loss function.
(3) We evaluate SARN on three real-world downstream prediction tasks over three real road networks. The results show that: (i) Compared with state-of-the-art self-supervised approaches, SARN achieves higher prediction accuracy in all three tasks over all three road networks, and the advantage is up to 34% (in the hit ratio for predicting similar trajectories). (ii) Compared with even supervised models for each task, SARN shows comparable results, and it outperforms them after fine-tuning on two of the tasks.

2 RELATED WORK

2.1 Road Network Embedding

Road network embedding methods can be divided into two categories: *task-specific embedding* and *task-agnostic embedding* (i.e., generic embedding).

Task-specific embedding. We use task-specific embeddings to denote road network embeddings learned in a supervised manner given a downstream task. Given supervision signals such as road type labels, shortest-path distances, or recommended routes, some studies [17, 28, 30, 33] embed intersections and road segments of a road network by generic neural network models such as *feedforward neural networks* (FFNs) and *graph neural networks* (GNNs), while others [16, 22, 40] design their own embedding models. All these studies encode information from the supervision signals into the learned embeddings. They do *not* necessarily encode the spatial structure of the road networks such as the connectivity (when using FFNs) and spatial locations of the road segments. The learned embeddings become inapplicable (or sub-optimal) when they are given to new tasks. *HRNR* [40] is the state-of-the-art road network embedding model in this category. It creates a three-level hierarchy of an input road network to capture its features at different granularity. By leveraging GNNs and two reconstruction tasks across the three levels, HRNR learns road segment embeddings that encode richer graph structure information. However, HRNR still requires task-dependant supervision signals for training. The road segment embeddings need to be learned for different tasks of interest separately.

Task-agnostic embedding. There are also road network embedding models that aim to learn generic embeddings without relying on any downstream tasks at training. *DeepWalk* [26] and *node2vec* [12] are two early self-supervised models for graphs which are adopted for road network embedding [20, 44, 46]. These methods generate node sequences by random walks on a graph, which are then treated as "sentences" and are used to learn vertex embeddings following the word2vec [24] word embedding idea. The learned embeddings encode local neighborhood structure of each vertex. Like GNNs, random walk-based embeddings suffer from lack of spatial structure of the road network.

A few other studies [11, 34, 35] devise their own models to learn generic embeddings. Specifically, Wang et al. [34] learn road intersection embeddings with an FFN that takes two intersections (in the form of one-hot embeddings) as inputs and predicts whether they are close and share the same intersection types. The first layer parameter weights of the trained FFN are used as the learning embeddings of the intersections. This model is further extended to learn embeddings for road segments and then trajectories [11, 35]. SRN2Vec [35], in particular, learns an FFN to predict whether two road segments are close and share the same road types. It considers the spatial proximity of the road segments but not the road network topology.

We aim to address the limitations above and learn *generic* road segment embeddings that encode *both spatial and topological information* with a self-supervised approach.

2.2 Graph Contrastive Learning

Contrastive learning is a self-supervised learning method that learns data representations based on *positive* (similar) and *negative* (dissimilar) data sample pairs. The aim is to generate data embeddings such that objects in a positive pair have much closer embeddings than those of objects in a negative pair.

Originated from computer vision [6, 8, 15], contrastive learning has been introduced to graph data, resulting in GCL models [14, 29, 42, 49] with strong learning capacity.

The basic idea of GCL is to learn vertex embeddings by maximizing the similarity between the embeddings of positive vertex pairs and minimizing that between the embeddings of negative vertex pairs. Different approaches have been proposed to guide the learning process. Some studies [14, 32] use an *instance-global discrimination* approach that maximizes the mutual information between the embeddings of a vertex and its corresponding sub-graph. In this case, a vertex and a sub-graph that contains the vertex forms a positive pair. Others [29, 42, 49] use *instance-instance discrimination* that learns from vertex pairs instead. We follow the latter approach and will detail a baseline algorithm using this approach in the next section. A comprehensive survey of GCL models can be found in Ref. [47].

Existing GCL models have focused on the generic graphs by capturing the topological structure of graphs. Applying them directly on road networks misses the spatial information such as the distances among road segments. Our SARN model addresses this limitation by incorporating spatial knowledge-based components.

3 PRELIMINARIES

We aim to learn road segment embeddings that encode rich topological and spatial structure information to enable fast and accurate query processing over road networks. We start with the data structure to represent a road network and our problem definition. Then, we present a direct adaptation of GCL for road segment embedding, which serves as the foundation of our SARN model. We also evaluate such method, i.e., SARN-w/o-MNL, as a baseline model in the experiments. We list frequently used symbols in Table 2.

Table 2: Frequently Used Symbols

Symbol	Description
G	An input road network graph
$\widetilde{G}, \widetilde{G}'$	Corrupted graph views of G
S	A set of road segments
$\mathbf{A^t}$	The adjacent matrix of G
$\mathbf{A^s}$	The spatial similarity matrix of G
s_i	The ith road segment, where $s_i \in S$
$\mathbf{h}_i$	The learned embedding of s_i

Input road network representation. We represent a road network as a *directed* graph $G = \langle S, \mathbf{A^t}, \mathbf{A^s} \rangle$. The set of graph vertices S represents the n road segments in the road network. This simplifies road segment embedding learning, following previous studies [17, 40]. Each vertex (i.e., a road segment) $s_i \in S$ is represented by the road type (e.g., motorway), length, radian, and coordinates of its start and end points, denoted by $\langle s_i.type, s_i.length, s_i.radian, s_i.start, s_i.end \rangle$. Here, the start and end points $s_i.start$ and $s_i.end$ further contain two features each, i.e., the latitude and longitude. The radian denotes the spatial direction of a road segment. We omit the road segment IDs so as to learn generic and ID-independent embeddings.

The adjacency matrix $\mathbf{A^t} \in \mathbb{R}^{n \times n}$ represents the topological connectivity between the road segments. A matrix element $\mathbf{A^t}_{i,j}$ is non-zero if s_j is directly connected from s_i, and 0 otherwise. We compute the value of $\mathbf{A^t}_{i,j}$ based on "weights" of s_i and s_j:

$$\mathbf{A^t}_{i,j} = \frac{1}{2} \cdot \big(\text{weight}(s_i) + \text{weight}(s_j)\big) \qquad (1)$$

Here, weight$(\cdot)$ returns the weight of a road segment, which is derived from its type, since the road types are a strong indicator of road importance. For example, motorways are more important than residential roads, as adding or removing a motorway is more likely to impact the overall network connectivity. We obtain the road types from OpenStreetMap [4] and empirically assign higher weights to roads of more important types, e.g., 6.0 for motorways and 2.0 for residential roads.

The *spatial similarity matrix* $\mathbf{A^s} \in \mathbb{R}^{n \times n}$ represents the similarity between the road segments based on their spatial properties. Since this is part of our contributions in the SARN model, we will detail it in the next section with SARN.

Problem definition. Given a road network G, we learn an embedding function $f : S \to \mathbb{R}^d$ that maps each road segment s_i in G to a vector $\mathbf{h}_i \in \mathbb{R}^d$, where d is a small constant.

We aim to achieve a function f that preserves road segment features and latent topological and spatial correlations among the road segments. The learned embeddings are expected to be applicable in a wide range of downstream tasks with or without fine-tuning, to produce accurate results efficiently.

A baseline GCL model. We present a baseline model that directly uses GCL for road network embedding with a *graph augmentation* module and a *graph encoding* module.

In every training epoch of a GCL model, the graph augmentation module corrupts G into two new graph variants, i.e., *graph views*, $\widetilde{G}$ and $\widetilde{G}'$. This can be done by randomly dropping vertices, dropping or adding edges, or masking vertex attributes.

The graph encoding module encodes each graph view with a separate GNN model. A nonlinear projection head further projects a GNN output embedding $\mathbf{h}_i$ to a lower dimensional embedding $\mathbf{z}_i \in \mathbb{R}^{d_z}$ ($d_z < d$) for loss computation.

Existing GCL models use the *InfoNCE loss* [25] for model learning as follows. Given an embedding $\mathbf{z}_i$ from graph view $\widetilde{G}$ that corresponds to vertex $s_i \in G$, the embedding $\mathbf{z}'_i$ of s_i that comes from the other graph view $\widetilde{G}'$ is considered a *positive sample* of $\mathbf{z}_i$. We further fetch K (a system parameter) *negative samples* of

$\mathbf{z}_i$, denoted by $\{\mathbf{z}_j^- \mid j \in [1, K]\}$, which are embeddings of random vertices from $\widetilde{G}'$. InfoNCE aims to maximize the similarity between $\mathbf{z}_i$ and $\mathbf{z}_i'$ and minimize that between $\mathbf{z}_i$ and $\mathbf{z}_j^-$:

$$\mathcal{L} = \mathbb{E}_{\mathbf{Z}}\Big[-\log \frac{\exp\big(\Lambda(\mathbf{z}_i, \mathbf{z}_i')/\tau\big)}{\exp\big(\Lambda(\mathbf{z}_i, \mathbf{z}_i')/\tau\big) + \sum_{j=1}^{K} \exp\big(\Lambda(\mathbf{z}_i, \mathbf{z}_j^-)/\tau\big)} \Big] \quad (2)$$

Here, $\mathbf{Z}$ denotes the distribution of $\mathbf{z}_i$, $\Lambda(\cdot,\cdot)$ measures the similarity between two embeddings, and τ is the *temperature* hyperparameter which controls the penalties on the negative samples that are more similar to $\mathbf{z}_i$.

When the GCL model is trained, the GNN model component is detached from the model and is used for producing embeddings for downstream tasks.

4 PROPOSED MODEL

Next, we detail the proposed *spatial-structure aware road network embedding* model, SARN. As Fig. 2 shows, given an input road network, SARN first constructs a graph representation $G = \langle S, \mathbf{A}^\mathbf{t}, \mathbf{A}^\mathbf{s} \rangle$. We detail how the *spatial similarity matrix* $\mathbf{A}^\mathbf{s}$ (Technical Contribution 1) is formed in Section 4.1. Then, G is corrupted into two graph views $\widetilde{G}$ and $\widetilde{G}'$ using a *spatial importance-based graph augmentation* module (Technical Contribution 2), which is detailed in Section 4.2. The graph views go through a graph encoding module that mainly consists of GNNs and nonlinear projection heads similar to the baseline GCL model above. We include the key computation steps of this module in Section 4.3 for completeness. This module outputs road segment embeddings from the two graph views. SARN uses a *spatial distance-based negative sampling* strategy (Technical Contribution 3) to sample from such embeddings to form positive and negative samples, as detailed in Section 4.4. The samples are fed into a *two-level loss function* (Technical Contribution 4) to compute the contrastive loss for model learning, which is detailed in Section 4.5.

4.1 Spatial Similarity Matrix

Given a road network, we first form its graph representation $G = \langle S, \mathbf{A}^\mathbf{t}, \mathbf{A}^\mathbf{s} \rangle$. The set of vertices S and the adjacency matrix $\mathbf{A}^\mathbf{t}$ have been described in Section 3. Here, we focus on the spatial similarity matrix $\mathbf{A}^\mathbf{s}$, which incorporates spatial correlations into the road network graphs, as defined below.

Definition 4.1 (Spatial similarity matrix). Given a road network graph $G = \langle S, \mathbf{A}^\mathbf{t} \rangle$, the spatial similarity matrix $\mathbf{A}^\mathbf{s} \in \mathbb{R}^{n \times n}$ represents the spatial similarity between road segments, where n denotes the number of road segments in G.

A matrix element $\mathbf{A^s}_{i,j}$ denotes the spatial similarity between s_i and s_j, which is the average of the *distance similarity* $\mathbf{A^s}_{i,j}.ds$ and the *angular similarity* $\mathbf{A^s}_{i,j}.as$:

$$\mathbf{A^s}_{i,j} = \begin{cases} \frac{1}{2}(\mathbf{A^s}_{i,j}.ds + \mathbf{A^s}_{i,j}.as), & i \neq j \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

$$\mathbf{A^s}_{i,j}.ds = \cos \frac{\pi \cdot \min\{\text{sp\_dist}(s_i,\ s_j),\ \delta_{ds}\}}{2\delta_{ds}} \quad (4)$$

$$\mathbf{A^s}_{i,j}.as = \cos \frac{\pi \cdot \min\{\text{ag\_dist}(s_i,\ s_j),\ \delta_{as}\}}{2\delta_{as}} \quad (5)$$

Here, we use the cosine functions to normalize the two similarity values into the same range of $[0, 1]$. Functions sp_dist$(\cdot,\cdot)$ and ag_dist$(\cdot,\cdot)$ return the spatial distance and the angular distance, respectively. Specifically, sp_dist$(\cdot,\cdot)$ computes the *haversine distance* [2] between the midpoints of two road segments, and ag_dist$(\cdot,\cdot)$ computes the absolute *angular distance* between two road segments ($|s_i.radian - s_j.radian|$). Two distance threshold parameters δ_{ds} and δ_{as} are used. They set the similarity values to 0 (since $\cos \frac{\pi}{2} = 0$) when the spatial distance and the angular distance are too large, respectively.

Our graph G has two types of edges, i.e., a directed *topological edge* $\overrightarrow{s_i, s_j}$ from s_i to s_j if $\mathbf{A^t}_{i,j} > 0$, and an undirected *spatial edge* $\overline{s_i, s_j}$ between s_i and s_j if $\mathbf{A^s}_{i,j} > 0$. In Fig. 2, there are nine directed road segments $s_1, s_2, \ldots, s_9$. Their topological and spatial edges are denoted by solid (blue) and dashed (red) lines, respectively. For example, s_7 shares topological edges with s_6 and s_8 (they share intersections), and it shares spatial edges with s_2 and s_4 (they are close and share similar directions).

The spatial edges (e.g., $\overline{s_2, s_7}$) capture the spatial similarity between the road segments. They help our model learn such information and encode it into the embeddings, even when the road segments are not directly connected in the original road network. This differs from the graph structures in typical GCL models where only topological edges are considered.

4.2 Spatial Importance-Based Graph Augmentation

Once G is formed, we need to augment it to produce two graph variants (i.e., graph views). Augmenting G to generate similar but not the same views is an important step towards high-quality embeddings [48, 49]. Next, we detail our augmentation strategy that considers the spatial importance of the road segments.

We generate two graph views $\widetilde{G}$ and $\widetilde{G}'$ by corrupting G before each training epoch, to produce the positive and negative samples. Since the positive samples require embeddings from $\widetilde{G}$ and $\widetilde{G}'$ that correspond to the same road segment $s_i \in S$, we only corrupt G by modifying its edges but not the vertices. Further, the embeddings of s_i from $\widetilde{G}$ and $\widetilde{G}'$ are supposed to be similar. Thus, $\widetilde{G}$ and $\widetilde{G}'$ should have a similar topology. This requires the edges with a stronger impact on the graph topology to have higher probabilities to be retained.

Inspired by the *GCA* model [49], we remove edges from G using probabilities based on their weights. Recall that the weights of the topological edges are based on the road segment importance, while the weights of the spatial edges are based on the importance to retain the spatial similarity relationship between two road segments. Thus, our graph augmentation strategy reflects the spatial importance of the road segments and their relationships.

We use two *corruption rate* parameters ρ_t and ρ_s to control the proportion of the topological edges and the spatial edges to be removed, respectively ($\rho_t, \rho_s \in (0, 1)$). We perform weighted sampling without replacement to remove edges.

For a topological edge $\overrightarrow{s_i, s_j}$, its probability of being sampled and removed (i.e., its *corruption probability*) is:

$$p(\overrightarrow{s_i, s_j}) = \sigma_\epsilon\Big(1 - \frac{\mathbf{A^t}_{i,j} - \min\{\mathbf{A^t}\}}{\max\{\mathbf{A^t}\} - \min\{\mathbf{A^t}\}}\Big) \quad (6)$$

Here, $\min\{\mathbf{A^t}\}$ and $\max\{\mathbf{A^t}\}$ denote the minimum and maximum non-zero elements of matrix $\mathbf{A^t}$, respectively; $\sigma_\epsilon(\cdot)$ is a linear function that maps the probability into range $[\epsilon, 1-\epsilon]$ to avoid corruption probabilities of 0 or 1, where ϵ is a small number. Intuitively, a topological edge with a larger weight connects two road segments with larger weights (i.e., more important road segments, cf. Eq. 1). It should be less likely to be removed (i.e., a lower corruption probability).

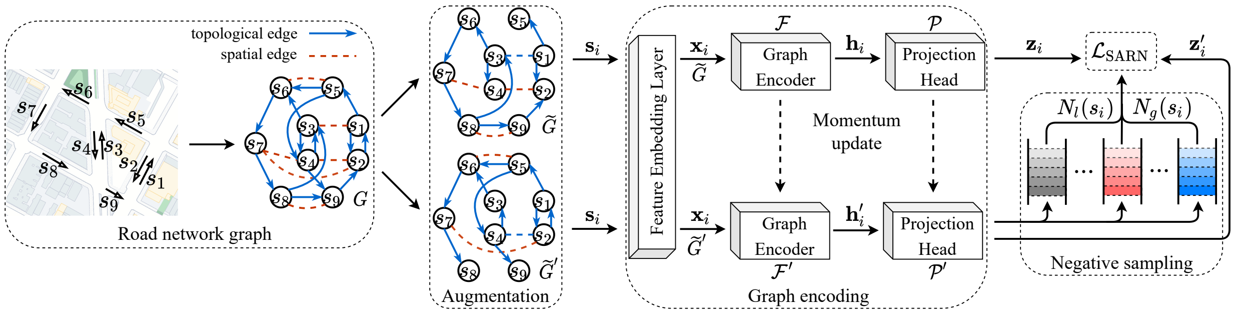


Figure 2: SARN model architecture

For a spatial edge $\overline{s_i, s_j}$, its corruption probability is:

$$p(\overline{s_i, s_j}) = \sigma_\epsilon\left(1 - \mathbf{A^s}_{i,j}\right) \tag{7}$$

We do not normalize $\mathbf{A^s}_{i,j}$ with the minimum and maximum elements of $\mathbf{A^s}$, because $\mathbf{A^s}_{i,j}$ is already in range $(0, 1)$ (cf. Eq. 3). Intuitively, a spatial edge with a larger weight denotes a higher spatial similarity between two road segments, which needs to be retrained to preserve the similarity relationship.

When there are both a topological edge and a spatial edge between s_i and s_j (we call them a "*dual-typed edge*" for short), we remove both edges when either is sampled. Such an edge removal strategy may cause the dual-typed edges being more likely to be removed, since they can be removed by either spatial or topological edge sampling. However, such edges are quite rare (e.g., 7.5% in the CD road network, cf. Table 3), and their impact is small as observed in our experiments.

4.3 Graph Encoding

Each corrupted graph view ($\widetilde{G}$ and $\widetilde{G}'$) is fed into a graph encoding module separately to learn embeddings for the graph vertices. The graph encoding module consists of a GNN and a nonlinear projection head. We follow the *momentum update* strategy [15] in our graph encoding module. Its key steps are briefed below for completeness of our SARN model.

Feature embedding layer. Before feeding $\widetilde{G}$ or $\widetilde{G}'$ into a GNN, the raw road segment features are first mapped to a higher dimensional space to enhance their representation power. This is done with a *feature embedding layer* shared by both corrupted graph views, as shown in Fig. 2.

As mentioned in Section 3, each road segment s_i is a 5-tuple with seven feature values in total, since there are two coordinates for each of the start and end points. We represent $s_i.type$ with an integer type ID (a one-hot vector). The other feature values are real numbers. We discretize each value domain with equi-sized bins (5 meters, 10 degrees, and 50 meters per bin for $s_i.length$, $s_i.radian$, and the end point coordinates, respectively) and convert each value into an integer bin ID (also a one-hot vector). We use $\mathbf{s}_i$ to denote the resultant feature vector.

Each feature value then goes through a separate linear embedding layer, due to its different value domain (and hence different amount of information to be encoded). The outputs of the linear layers for all seven feature values are concatenated to form a vector $\mathbf{x}_i \in \mathbb{R}^{d_f}$ which is used as the input for the next module, where d_f is a system parameter.

Graph encoder. The mapped corrupted graphs $\widetilde{G}$ and $\widetilde{G}'$ (with mapped feature vectors $\mathbf{x}_i$) then go through GNNs $\mathcal{F}$ and $\mathcal{F}'$ separately (cf. Fig. 2). We use the *graph attention network* (GAT) [31] as the GNN. [1] Models $\mathcal{F}$ and $\mathcal{F}'$ each learns the embedding $\mathbf{h}_i \in \mathbb{R}^d$ of a vertex s_i by aggregating those of its (first-order) neighboring vertices using the *self-attention mechanism* (readers familiar with GATs may skip this paragraph):

$$\mathbf{h}_i = \Big\Vert_{l=1}^{L} \sigma\left(\sum_{s_j \in \mathcal{N}_i} \alpha^l_{ij} \mathbf{W}^l \mathbf{x}_j\right) \tag{8}$$

Here, $\mathcal{N}_i$ denotes the set of neighboring vertices of s_i. We consider neighboring vertices connected to s_i by either a topological edge or a spatial edge, such that the learned embedding of s_i encodes both the topological and spatial structure information. Function σ is a nonlinear activation function (ELU), and $\Vert$ denotes concatenation (averaging is used for the final GAT layer – we use three layers). There are L (a hyper-parameter, 4 in our model) sets of independent attention weights each denoted by α^l and $\mathbf{W}^l$ (i.e., *multi-head attention*), to represent the vertex correlations.

To compute the *attention coefficient* α_{ij} of $\mathbf{x}_i$ and $\mathbf{x}_j$, both $\mathbf{x}_i$ and $\mathbf{x}_j$ are first mapped to $\mathbb{R}^d$ via multiplying by a learned and shared weight matrix $\mathbf{W} \in \mathbb{R}^{d \times d_f}$. The mapped vectors are concatenated and then further mapped back into a single real value $e_{ij} \in \mathbb{R}$ by a learned weight vector $\mathbf{a} \in \mathbb{R}^{2d}$:

$$e_{ij} = \mathbf{a}^T \cdot [\mathbf{W}\mathbf{x}_i \Vert \mathbf{W}\mathbf{x}_j] \tag{9}$$

A *softmax*-like function is applied to normalize the attention coefficients among the neighbouring vertices.

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(e_{ij}))}{\sum_{s_k \in \mathcal{N}_i} \exp(\text{LeakyReLU}(e_{ik}))} \tag{10}$$

We use the embedding $\mathbf{h}_i$ learned via the graph encoder $\mathcal{F}$ as the embedding of s_i in downstream tasks.

Projection head. An output embedding of a GNN, i.e., $\mathbf{h}_i$, is fed into a projection head, which is an FFN that maps $\mathbf{h}_i$ to a lower dimensional vector $\mathbf{z}_i \in \mathbb{R}^{d_z}$ ($d_z < d$) before computing the loss. This step was shown to be effective to improve the quality of the learned embeddings [8, 9]. Let FC and ReLU be a fully connected layer and the ReLU activation function, respectively, and '∘' denote function composition.

$$\mathbf{z}_i = (\text{FC} \circ \text{ReLU} \circ \text{FC})(\mathbf{h}_i) \tag{11}$$

We denote the two projection heads used for the two corrupted graph views by $\mathcal{P}$ and $\mathcal{P}'$, respectively.

Weight update. Each of the GATs $\mathcal{F}$ and $\mathcal{F}'$ and the projection heads $\mathcal{P}$ and $\mathcal{P}'$ has its own weight values. The weights of $\mathcal{F}$ and $\mathcal{P}$ are updated (i.e., learned) by gradient descent with the contrastive loss (detailed in Section 4.5) computed for each mini-batch input. Then, the weights of $\mathcal{F}'$ and $\mathcal{P}'$ are updated

[1]We use GAT to learn unified underlying correlation weights which can imply both topological and spatial structure-based correlation between vertices based on the input features of vertices and their neighbouring relationships (which can be obtained from (0,1)-adjacency matrices of $\mathbf{A^t}$ and $\mathbf{A^s}$). Comparing with using $\mathbf{A^t}$, $\mathbf{A^s}$, or their sum, the weights as learned one by the attention mechanism of GAT have better expressive power that can model both types of weights adaptively.

by *momentum update* [15]:

$$\mathbf{W}_{\mathcal{X}'} = m\mathbf{W}_{\mathcal{X}'} + (1-m)\mathbf{W}_{\mathcal{X}} \tag{12}$$

Here, $\mathbf{W}_{\mathcal{X}}$ denotes all weights of model $\mathcal{X}$, where $\mathcal{X} \in \{\mathcal{F}, \mathcal{P}\}$. $m \in (0,1)$ is the *momentum coefficient* (a hyper-parameter) that controls the stableness of the weight updates in $\mathcal{F}'$ and $\mathcal{P}'$. We use a large value (0.999 in our experiments) for m to keep stable weight updates. This enables reusing embeddings learned from previous mini-batches for the next round of negative sampling, thus enlarging the pool of negative samples.

4.4 Spatial Distance-Based Negative Sampling

By now, we have obtained vertex embeddings of both graph views $\widetilde{G}$ and $\widetilde{G}'$ from the graph encoding module. The next step is to compute a training loss which will be used for model parameter updates in the backpropagation process. Before introducing our loss function, we need to first describe our procedure to sample the vertex pairs for the loss computation.

Similar to the baseline GCL model in Section 3, SARN needs both positive and negative samples to guide model training. We adopt the same positive sampling strategy used in the baseline GCL model, i.e., given a vertex s_i from $\widetilde{G}$, (the embedding of) the same vertex from $\widetilde{G}'$ is used as a positive sample for s_i .

Next, we focus on negative samples (cf. Eq. 2), which are critical to provide learning signals in GCL since there is no external supervision signals. Existing GCL models treat all vertices (road segments) in $\widetilde{G}'$ equally and sample from them randomly to form the negative samples. As discussed earlier, road segments of different distances from a target road segments contribute differently to the contrastive learning process. We thus propose a spatial distance-based negative sampling strategy.

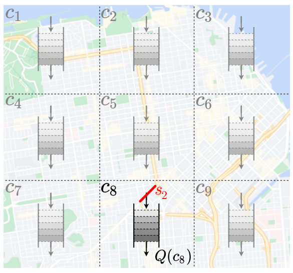


Figure 3: Spatial distance-based negative sampling

Our sampling strategy uses a grid partitioning on the space of the road network G, where each cell c_i has a side length of *clen*. Each cell c_i maintains a queue $Q(c_i)$ of size ϕ. The queue stores the last ϕ embeddings $\mathbf{z}'_j$ produced by projection head $\mathcal{P}'$ (in previous mini-batches, following *MoCo* [15]) that correspond to road segments s_j whose midpoints fall in c_i. For example, Fig. 3 shows a road network space with 3×3 grid. When $\mathcal{P}'$ produces embedding $\mathbf{z}'_2$ of s_2, we push $\mathbf{z}'_2$ into $Q(c_8)$, since s_2 is in c_8.

Inspired by MoCo [15], we use queues to store embedding samples from the recent mini-batches, where the stored embeddings will be used as negative samples during the training stage. Such a technique can offer a larger set of negative samples than those in contrastive learning models [8, 42] that sample negative samples from only one mini-batch. A large set of negative samples helps learn more robust representations since it prompts the distribution of embeddings with uniformity [38].

Using the grid partition, we can obtain two types of negative samples, namely *local negative samples* and *global negative samples*. Given a target road segment s_i, let $s_i.cell$ be the cell enclosing the midpoint of s_i, and $Q(s_i.cell)$ be its queue. The set of local negative samples of s_i contains the embeddings in $Q(s_i.cell)$ (except those corresponding to s_i), denoted by:

$$N_l(s_i) = \{\mathbf{z}_j | \mathbf{z}_j \in Q(s_i.cell), s_j \neq s_i\} \tag{13}$$

Here, $\mathbf{z}_j$ is the embedding corresponding to s_j. Such samples help differentiate road segments in the same local area.

The set of global negative samples of s_i is generated from the embeddings in the queues of the other cells. They provide training signals to differentiate road segments in different areas. Since vertices in the other cells are far away from s_i, it is unnecessary to learn fine-grained differences between their embeddings and that of s_i. We thus leverage a *readout* function (we use the mean aggregate) $\mathcal{R}(\cdot)$ to aggregate the embeddings in each queue to obtain their overall representation. The resultant aggregated embedding is also in $\mathbb{R}^{d_z}$ and is used as a global negative sample. The set of global negative samples is thus denoted by:

$$N_g(s_i) = \{\mathcal{R}(Q(c_k)) | c_k \in C, c_k \neq s_i.cell\} \tag{14}$$

Here, C denotes the set of all cells in the grid.

4.5 Two-level Loss Function

Finally, we introduce our loss function, which has two loss terms, *local contrastive loss* and *global contrastive loss*, based on the local and the global negative samples. Intuitively, the global contrastive loss guides our model SARN to learn a general pattern shared by all road segments in the same cell, which is different from cell to cell. Meanwhile, the local contrastive loss further guides SARN to learn more subtle differences among the embeddings of the road segments in the same cell.

Local contrastive loss. The local contrastive loss $\mathcal{L}_l(s_i)$ for a target road segment s_i is similar to the InfoNCE loss (cf. Eq. 2), except that we use the local negative samples as $\mathbf{z}_j^-$ instead of random ones:

$$\mathcal{L}_l(s_i) = -\log \frac{\exp\big(\Lambda(\mathbf{z}_i, \mathbf{z}'_i)/\tau\big)}{\exp\big(\Lambda(\mathbf{z}_i, \mathbf{z}'_i)/\tau\big) + \sum\limits_{\mathbf{z}_j^- \in N_l(s_i)} \exp\big(\Lambda(\mathbf{z}_i, \mathbf{z}_j^-)/\tau\big)} \tag{15}$$

Here, $\mathbf{z}_i$ and $\mathbf{z}'_i$ denote the embeddings of s_i produced by $\mathcal{P}$ and $\mathcal{P}'$, respectively. Function $\Lambda(\cdot,\cdot)$ measures the similarity of two embeddings (we use their dot product) and τ is the temperature parameter as in Eq. 2.

Global contrastive loss. The global contrastive loss $\mathcal{L}_g(s_i)$ for s_i is computed using the global negative samples as $\mathbf{z}_j^-$:

$$\mathcal{L}_g(s_i) = -\log \frac{\exp\big(\Lambda(\mathbf{z}_i, \mathbf{z}_i^+)/\tau\big)}{\exp\big(\Lambda(\mathbf{z}_i, \mathbf{z}_i^+)/\tau\big) + \sum\limits_{\mathbf{z}_j^- \in N_g(s_i)} \exp\big(\Lambda(\mathbf{z}_i, \mathbf{z}_j^-)/\tau\big)} \tag{16}$$

Here, $\mathbf{z}_i^+$ denotes the aggregated embedding of $s_i.cell$, i.e., $\mathbf{z}_i^+ = \mathcal{R}(Q(s_i.cell))$, which is used as the positive sample in the global contrastive loss.

We combine the two loss terms with a trade-off parameter $\lambda \in (0,1)$ to obtain our final loss function $\mathcal{L}_{\text{SARN}}$:

$$\mathcal{L}_{\text{SARN}} = \mathbb{E}_{s_i \in S^*}\big[\lambda \mathcal{L}_l(s_i) + (1-\lambda)\mathcal{L}_g(s_i)\big] \tag{17}$$

Here, S^* denotes the set of road segments in a mini-batch.

Training algorithm. Algorithm 1 summarizes the training process of SARN. In each epoch, we generate two corrupted graph views $\widetilde{G}$ and $\widetilde{G}'$ (Lines 2 and 3, Section 4.2) and use them in every mini-batch with a different subset S^* of the set S of all road segments. For each subset S^*, we compute the graph embeddings (Lines 5 and 6) which are fed into the projection heads (Lines 7 and 8). Then, we leverage the local and the global negative samples (denoted as **N**) and the projection head outputs for loss computation (Lines 9 and 10), the result of which is used to guide weight updates (Lines 11 to 14). We update the queues of the cells with the newly computed embeddings before the next mini-batch is processed (Line 15). The algorithm returns the output **H** of graph encoder $\mathcal{F}$ as the road network embeddings when the training completes.

Algorithm 1: SARN_training

```
Input: G: road network graph;
       $\rho_t, \rho_s, \lambda, m$: model parameters.
Output: H: road segment embeddings
1  while not converged do
2      $\widetilde{G} \leftarrow$ augment_graph($G, \rho_t, \rho_s$);
3      $\widetilde{G}' \leftarrow$ augment_graph($G, \rho_t, \rho_s$);
4      for $S^* \subset S$ do
5          $\mathbf{H} \leftarrow \mathcal{F}(S^*, \widetilde{G})$;
6          $\mathbf{H}' \leftarrow \mathcal{F}'(S^*, \widetilde{G}')$;
7          $\mathbf{Z} \leftarrow \mathcal{P}(\mathbf{H})$;
8          $\mathbf{Z}' \leftarrow \mathcal{P}'(\mathbf{H}')$;
9          $\mathbf{N} \leftarrow$ fetch_negative_samples($S^*$);
10         $\mathcal{L}_{\text{SARN}} \leftarrow$ compute_loss($\mathbf{Z}, \mathbf{Z}', \mathbf{N}, \lambda$);
11         $\mathcal{W}_{\mathcal{F}} \leftarrow$ gradient_descent($\mathcal{W}_{\mathcal{F}}, \mathcal{L}_{\text{SARN}}$);
12         $\mathcal{W}_{\mathcal{P}} \leftarrow$ gradient_descent($\mathcal{W}_{\mathcal{P}}, \mathcal{L}_{\text{SARN}}$);
13         $\mathcal{W}_{\mathcal{F}'} \leftarrow$ momentum_update($\mathcal{W}_{\mathcal{F}}, \mathcal{W}_{\mathcal{F}'}, m$);
14         $\mathcal{W}_{\mathcal{P}'} \leftarrow$ momentum_update($\mathcal{W}_{\mathcal{P}}, \mathcal{W}_{\mathcal{P}'}, m$);
15         update_queue($\mathbf{Z}'$);
16 return H;
```

Model costs. Each epoch takes $O(n_e) + \mathbb{M}(n) + O(K \cdot n \cdot d^2)$ time. Here, $O(n_e)$ denotes the time to generate the graph views, assuming n_e edges in G. Cost $\mathbb{M}(n)$ denotes the time to go through the graph encoding module and generate the projected embeddings for all n vertices, which is determined by the time costs of the neural network models used. We generalize it to $\mathbb{M}(n)$ for simplicity. Cost $O(K \cdot n \cdot d^2)$ denotes the time for loss computation over n vertices, assuming K negative samples for each vertex, i.e., $K = |N_l(s_i)| + |N_g(s_i)|$. In comparison, our baseline GCL models *GCA* [49] and *GraphCL* [42] (detailed in the next section) share the same time costs $O(n_e) + \mathbb{M}(n)$ with SARN in graph view and embedding generation. For loss computation, GCA takes $O(n^2 \cdot d^2)$ time while GraphCL takes $O(|S^*| \cdot n \cdot d^2)$ time, since they use all vertices and only other vertices in the same mini-batch as the negative samples, respectively.

SARN takes $O(n_e)$ memory space at training which is dominated by the size of the adjacency matrix and the spatial similarity matrix. We use an edge list-based implementation of these matrices to reduce the space usage. The trained model takes $O(n \cdot d)$ space to store, for the learned embeddings and the model parameters. These costs are the same as those of GCA and GraphCL.

5 EXPERIMENTS

We show the applicability of the SARN embeddings via three downstream tasks: *road property prediction*, *trajectory similarity prediction*, and *shortest-path distance prediction*. These three tasks focus on individual road segments, sequences of road segments, and relative position of a pair of road segments, respectively. We use these tasks as they relate to the topological and spatial structures of the road networks, which are the focus of this study. Tasks that involve other contextual factors such the time (e.g., travel time predictions) are beyond the scope of this study and will be considered in future work.

5.1 Experimental Setup

Datasets. We use three road network datasets that are extracted from OpenStreetMap [3]. They correspond to a region within the Second Ring Road of Chengdu (**CD**, a capital city in southwest China), a region within the Second Ring Road of Beijing (**BJ**, the capital of China), and a northeastern region of San Francisco (**SF**, USA). We build a road network graph G for each dataset. The three datasets contain 29,593, 36,809, and 37,284 road segments, respectively. The datasets are summarized in Table 3. Besides, we also extract another two road networks with different sizes in San Francisco, which are used to study the impact of the road network size along with SF. The details will be discussed in Section 5.2.4.

Table 3: Road Network Datasets

	CD	**BJ**	**SF**
Number of road segments	29,593	36,809	37,284
Number of edges in $\mathbf{A}^{\mathbf{t}}$	50,325	66,598	60,410
Number of edges in $\mathbf{A}^{\mathbf{s}}$	48,002	63,875	59,606
Area (km^2)	10.13×11.26	9.49×8.74	5.72×5.69

For the trajectory similarity prediction task, we use three real-world trajectory datasets, i.e., **DiDi** [1], **T-Drive** [45], and **SF-Cab** [27]. DiDi contains ride-hailing vehicle trajectories recorded during November 2016 (first seven days) in Chengdu. T-Drive and SF-Cab contain weekly and monthly taxi trajectories recorded in Beijing and San Francisco, respectively. We break each trajectory into two when the time interval between two adjacent GPS points exceeds 20 minutes. We also remove any trajectory points outside the regions of the CD, BJ, and SF road networks. Then, we randomly sample 10k trajectories from each trajectory dataset and map them to the CD, BJ, and SF road networks using a map-matching algorithm [23]. By default, we truncate the matched trajectories to a maximum length of 60 segments to form the trajectory datasets. We also study the impact of number of segments by varying it from 60 to 180 in the experiments. Here, we focus on the number of segments instead of the physical trajectory length. This is because the neural network models for trajectory modelling take input in the form of sequence of segments, and their learning performance is impacted more by the number of segments than by the physical trajectory length (e.g., a lengthy trajectory can be represented by limited number of segments using trajectory simplification algorithms [10]).

Competitors. We compare embeddings learned by our SARN model with those learned by four *self-supervised* embedding models, one *task-agnostic supervised* model and two *task-specific supervised* models.

- Self-supervised models:
  1. **node2vec** [12] is a widely used self-supervised graph embedding model that applies random walks on a graph to generate sequences of vertices (i.e., road segments), which are then treated as "sentences" and are used for embedding learning with the *word2vec* technique [24].
  2. **GraphCL** [42] is a representative GCL model that follows the description of the baseline model in Section 3, except that its graph encoding modules for both graph views share parameters. It uses road segments in the same mini-batch of the target segment as negative samples.
  3. **GCA** [49] extends GraphCL by an *adaptive graph augmentation* based on the weights of the vertex attributes and edges. A vertex or an edge of a higher weight is more likely to be retained in the augmented graph views. GCA uses all other vertices in G to form negative samples.

(4) **SRN2Vec** [35] learns an FFN that maps road segments which are spatially close and are of the same type to be close in the embedding space (this can also be seen as a self-supervised model), as described in Section 2.1.

- Task-agnostic supervised model:

(1) **HRNR** [40] is the state-of-the-art task-agnostic supervised road network embedding model. It creates a hierarchical road network representation and uses two reconstruction tasks to learn the inter-relationships between the three layers of the road network hierarchy. While this model does not rely on a particular downstream task, it needs to be connected with a task-specific component to obtain supervision signals for embedding learning.

- Task-specific supervised models:

(1) **NEUTRAJ** [41] is a recent model for trajectory similarity prediction. It uses a *recurrent neural network* (RNN) with a spatial attention memory unit to learn trajectory representation and predict trajectory similarity.
(2) **RNE** [16] is the state-of-the-art for shortest-path distance prediction with high efficiency. It uses FFNs to learn vertex embeddings based on the road network hierarchy and predict shortest-path distances.

We use the released code for all these competitors except SRN2Vec which does not have released code. We implemented SRN2Vec following its proposal [35]. We use the recommended parameters of these models provided in the associated papers unless specified otherwise.

Implementation details. We use the same backbone graph encoder GAT for GraphCL, GCA and SARN for fair comparison. The embedding size d is 128 for all models. For SARN, we set the distance thresholds δ_{ds} and δ_{as} to 200 meters and $\frac{\pi}{8}$, respectively. We use 0.4 as the default edge corruption rates ρ_t and ρ_s. We use $1,000$ (this value bounds K) as the total size of all sample queues on each road network. This means that the cell side length *clen* and queue size ϕ are 1,200 meters and 11 for CD, 1,200 meters and 16 for BJ, and 600 meters and 10 for SF, respectively.

We use the Adam optimizer with early stopping in training. The maximum number of training epochs of SARN is 200 and the patience is 20. The learning rate is initialized to 0.005 and it decays following the cosine annealing schedule. We use a mini-batch size of 128. In the loss function (Eq. 17), λ is set to 0.4 with the temperature $\tau = 0.05$. The experimental results of parameter study can be found in Section 5.5.

We implement SARN with PyTorch 1.8.1. All experiments are run on a 64-bit Linux server with an Intel Xeon Gold 6132 CPU, 64 GB RAM, and an NVIDIA Tesla V100 GPU. We repeat the experiments 5 times with different random seeds and report the average performance.

5.2 Performance on Downstream Tasks

We first investigate the effectiveness of SARN on the downstream tasks. Even though our embeddings provide prediction results with both high accuracy and efficiency, we focus on the accuracy in the following discussion. This is because the efficiency of embedding-based approaches have been shown in the baseline models NEUTRAJ and RNE in each task. We retain the efficiency by following prediction procedures similar to these models.

For the self-supervised models node2vec, GraphCL, GCA, SRN2Vec, and our model SARN, we first learn the embedding independently from any downstream tasks. Then, for each downstream task, we learn a simple prediction model (detailed in each task), where the learned embeddings are frozen.

For HRNR, the embeddings and prediction models are learned together in an end-to-end fashion for each task. NEUTRAJ and RNE are both end-to-end supervised methods. We directly use them for their respective target tasks.

Additionally, we freeze the RNE embeddings and used them with the prediction models for the other two tasks like those of the self-supervised models. We do not do this with NEUTRAJ because it does not produce road segment embeddings.

We further implement a fine-tuned version of SARN, denoted by **SARN***, to observe the quality of the embeddings when SARN receives task-specific supervision signals. SARN* is initialized to a trained SARN model, and its final layer of the graph encoder $\mathcal{F}$ is fine-tuned together with training the prediction model of each downstream task.

5.2.1 Road Property Prediction.

Setup. Following previous studies [35, 40], we use the learned embeddings to predict the properties of each road segment. The intention here is to show the effectiveness of the learned embeddings to differentiate road segments of different properties. The OpenStreetMap datasets contain a variety of road properties. We predict the speed limits of road segments here as they are *not* part of the input features of SARN. There are 613, 57, and 7,283 road segments in CD, BJ, and SF datasets with speed limit values, respectively (we use a 6:2:2-spit for training, validation, and testing). The numbers of different speed limits in the three datasets are 7, 4, and 10, respectively.

We note that the road type (which is one of the model input features) and the speed limit have some correlation naturally. However, this correlation is not always high. The normalized mutual information (NMI) between the two features are 0.80, 0.73, and 0.39 for the CD, BJ, and SF datasets, respectively. The NMI for the largest dataset SF, in particular, is quite low, i.e., 0.39, which shows that this prediction problem is still non-trivial.

We use an FFN with one hidden layer of 32 nodes as the classifier. We report the F1 score and the *area under the receiver operating characteristic curve* (AUC) score.

Results. Table 4 presents the results. Overall, SARN outperforms all self-supervised models on all three datasets, even though the baseline models have very strong results already. Comparing with GCA which is the more recent GCL model (and also the best self-supervised baseline), SARN improves by up to 3.78% on F1 score on SF. Note that this improvement is more substantial than what GCA has achieved over GraphCL, i.e., up to 0.3%. The advantage of SARN confirms the effectiveness of using spatial knowledge to further enhance the GCL models. On BJ, GraphCL and GCA have the same performance, which is a coincidence on this small dataset.

After fine-tuning with task-specific supervision signals, SARN* outperforms SARN on CD and SF as expected. SARN* even outperforms the supervised model HRNR, by up to 3.65% in F1 on SF. This result is somewhat unexpected as SARN* is only fine-tuned but not trained from scratch with task-specific supervision signals like HRNR is. We attribute the advantage to the spatial knowledge encoded in the SARN embeddings, which helps identify road segments of similar speed limits (e.g., parallel ring roads or roads in the same region with similar speed limits). RNE is also outperformed by SARN* as it was not designed for the task.

Table 4: Road Property Prediction Results

(SARN and SARN* results are in **bold**. The best self-supervised and supervised baseline results are underlined. "Gain (%)" denotes the relative improvement of SARN over the best self-supervised baseline, and that of SARN* over the best supervised baseline respectively. Same for the tables below.)

Category	Method	CD		BJ		SF	
		F1 (%)	AUC (%)	F1 (%)	AUC (%)	F1 (%)	AUC (%)
Self-supervised	node2vec	89.11±1.96	93.64±1.14	90.00±13.69	93.33±9.13	60.99±1.55	78.33±0.86
	SRN2Vec	74.31±1.58	85.01±0.92	46.67±12.64	64.44±8.43	54.85±0.51	74.92±0.28
	GraphCL	96.75±0.81	98.10±0.47	98.33±3.73	98.89±2.48	87.82±2.28	93.24±1.26
	GCA	97.07±1.58	98.29±0.92	98.33±3.73	98.89±2.48	87.89±1.91	93.27±1.06
	SARN	**98.70±0.73**	**99.24±0.42**	**100.00±0.00**	**100.00±0.00**	**91.21±0.71**	**95.12±0.40**
	Gain (%)	+1.68	+0.97	+1.69	+1.12	+3.78	+1.98
Fine-tuned	**SARN***	**99.02±1.34**	**99.43±0.78**	**100.00±0.00**	**100.00±0.00**	**92.68±1.36**	**95.94±0.75**
Supervised	HRNR	95.93±2.37	97.63±1.38	81.67±19.00	87.78±12.67	89.42±0.86	94.12±0.48
	RNE	97.40±1.06	98.48±0.62	98.33±3.73	98.89±2.48	87.71±0.82	93.17±0.46
	Gain (%)	+1.67	+0.96	+1.69	+1.12	+3.65	+1.93

Table 5: Trajectory Similarity Prediction Results

Category	Method	CD			BJ			SF		
		HR@5 (%)	HR@20 (%)	R5@20 (%)	HR@5 (%)	HR@20 (%)	R5@20 (%)	HR@5 (%)	HR@20 (%)	R5@20 (%)
Self-supervised	node2vec	43.03±1.89	60.44±1.20	81.91±1.41	26.45±3.61	50.82±6.06	65.21±8.42	25.42±0.91	41.35±1.00	55.53±1.72
	SRN2Vec	59.87±1.33	73.16±0.56	95.22±0.55	48.51±0.85	68.66±0.88	89.06±1.34	59.73±0.94	72.65±0.50	94.29±0.39
	GraphCL	58.09±2.52	72.23±2.20	94.06±1.64	39.91±1.60	61.78±1.15	81.47±1.25	42.42±2.65	57.99±2.80	81.33±3.19
	GCA	55.97±3.13	71.98±1.97	94.75±1.24	41.64±3.34	62.97±2.47	81.99±2.57	52.48±2.36	66.24±1.60	90.92±1.16
	SARN	**66.42±1.29**	**78.44±0.77**	**98.36±0.32**	**65.18±1.92**	**79.73±1.15**	**98.22±0.70**	**68.80±0.92**	**78.64±0.35**	**98.39±0.18**
	Gain (%)	+10.94	+7.22	+3.30	+34.36	+16.13	+10.28	+15.19	+8.24	+4.35
Fine-tuned	**SARN***	**71.69±2.38**	**81.27±1.98**	**99.32±0.37**	**70.54±1.88**	**82.98±1.33**	**99.22±0.49**	**75.67±0.40**	**82.33±0.51**	**99.48±0.13**
Supervised	HRNR	67.33±3.57	78.47±2.53	97.55±0.88	64.43±1.49	79.36±0.86	97.18±0.40	65.50±0.89	76.39±0.65	95.54±0.79
	NEUTRAJ	72.18±1.66	79.79±0.76	99.18±0.35	70.94±0.82	78.79±0.38	98.81±0.20	72.94±2.40	79.40±1.77	98.61±0.67
	RNE	68.27±2.13	79.17±1.62	98.26±0.59	66.19±0.84	79.63±0.31	98.08±0.23	69.59±3.73	78.13±2.80	98.16±1.13
	Gain (%)	-0.68	+1.86	+0.14	-0.58	+5.31	+0.41	+3.74	+3.69	+0.88

Table 6: Shortest-Path Distance Predication Results (Smaller values are better.)

Category	Method	CD		BJ		SF	
		MRE (%)	MAE (meter)	MRE (%)	MAE (meter)	MRE (%)	MAE (meter)
Self-supervised	node2vec	52.76±2.11	2017.71±95.36	57.36±1.45	2054.52±49.34	59.59±3.64	1196.62±162.91
	SRN2Vec	57.19±1.08	2208.95±57.84	60.16±0.55	2183.24±12.97	62.94±0.32	1188.38±4.55
	GraphCL	15.35±1.02	656.96±34.53	16.48±3.16	678.92±117.41	12.31±1.84	280.99±41.39
	GCA	11.88±2.56	530.23±100.50	9.81±1.18	434.01±53.58	8.76±0.47	210.70±8.83
	SARN	**10.19±0.25**	**462.46±11.31**	**7.44±0.37**	**324.12±15.84**	**7.53±0.12**	**174.08±2.79**
	Gain (%)	+14.23	+12.78	+24.16	+25.32	+14.04	+17.38
Fine-tuned	**SARN***	**9.37±0.36**	**401.77±26.56**	**7.08±0.12**	**313.34±9.24**	**7.21±0.28**	**174.24±12.37**
Supervised	HRNR	7.08±0.39	303.30±31.40	5.36±0.24	205.26±17.05	4.85±0.34	98.65±9.21
	RNE	9.55±0.15	462.11±16.53	7.63±0.15	302.87±7.28	7.94±0.13	176.01±2.83
	Gain (%)	-32.31	-32.46	-31.98	-52.66	-48.65	-76.63

5.2.2 *Trajectory Similarity Prediction.*

Setup. Next, we show the effectiveness of the SARN embeddings in differentiating a series of road segments (i.e., trajectories). Following the baseline method NEUTRAJ [41], we use the embeddings of trajectories to predict their similarities, and we compute the top-k most similar trajectories based on the predicted similarity. This procedure helps reduce the time needed to compute the similarity between two trajectories (by avoiding pairwise point distance computation), and hence it reduces the time for finding the top-k most similar trajectories as shown by NEUTRAJ [41].

Given a trajectory, we feed the road segments (i.e., embeddings) into a 2-layer GRU model to learn a trajectory embedding (a 1024-dimensional vector) and we use the last hidden state of GRU as the trajectory embedding. We compute the L_1 distance of two trajectory embeddings as the predicated distance of the two trajectories. For model training and testing, we compute the *Fréchet* distance [5] between two trajectories as the ground truth, which is frequently used in trajectory query studies [19, 37, 39]. It is straightforward to replace it with another metric. We use a 6:2:2-split on each dataset for training, validation, and testing.

Following NEUTRAJ, we report *HR@5*, *HR@20*, and *R5@20*. HR@k ($k = 5, 20$) is the percentage of ground truth top-k trajectories in the predicted top-k results. *R5@20* is the percentage of ground truth top-5 trajectories in the predicted top-20 results.

Results. Table 5 shows the results. SARN again outperforms all self-supervised competitors, and the advantage now becomes larger as the task becomes more complex, i.e., up to 34.36% in HR@5 on BJ, while SARN* further improves over SARN. The large advantage of our models can be explained by their encoding of the road segment angles and spatial locations, which are core to trajectory similarity measurements. This also explains for the strong performance of SRN2Vec as it also encodes the spatial structure of the road segments.

SARN* achieves comparable performance to that of the recent trajectory similarity prediction model NEUTRAJ, where NEUTRAJ is 0.68% better in HR@5 on CD while SARN* is 3.74% better in HR@5 on SF. Note that NEUTRAJ is a highly optimized model for trajectory similarity prediction only. It does not produce road segment embeddings that can be used in other applications, while our models do.

The performance gap between HRNR and SARN* now becomes larger, as this task focuses more on the spatial structure, which is not considered by HRNR. RNE again shows strong results (although still worse than our models). We conjecture that this is because it learns pairwise distances of all road segments, which essentially encodes the entire graph structure. This can be helpful in different tasks.

Table 7: Impact of the Number of Road Segments in Trajectories on Trajectory Similarity Prediction.

Metric	Method	60	120	180
HR@5 (%)	SRN2Vec	48.51±0.85	15.87±2.13	12.15±0.83
	SARN	**65.18±1.92**	**31.88±2.00**	**27.58±1.87**
	SARN*	**70.54±1.88**	**43.00±1.63**	**41.92±1.35**
	NEUTRAJ	70.94±0.82	42.92±1.00	40.97±0.81
HR@20 (%)	SRN2Vec	68.66±0.88	33.19±2.48	27.27±1.84
	SARN	**79.73±1.15**	**50.40±2.77**	**44.32±2.34**
	SARN*	**82.98±1.33**	**63.26±1.10**	**59.88±1.52**
	NEUTRAJ	78.79±0.38	62.37±0.94	59.88±1.03
R5@20 (%)	SRN2Vec	89.06±1.34	41.95±3.61	32.97±2.43
	SARN	**98.22±0.70**	**67.58±4.18**	**58.71±3.88**
	SARN*	**99.22±0.49**	**84.54±1.33**	**82.13±2.69**
	NEUTRAJ	98.81±0.20	83.85±1.22	80.52±1.58

Impact of the number of road segments. We further study the impact of the number of road segments in each trajectory by varying the maximum number of road segments allowed for a trajectory when truncating the trajectories in each dataset. We show the results when the maximum number of road segments increases from 60 to 180 on the T-Drive dataset in Table 7, since the trajectories in T-Drive have the largest physical length (i.e., up to 19,418 meters for trajectories of 180 segments).

For conciseness, we show results of our models and the best self-supervised and supervised baselines, i.e., SRN2Vec and NEUTRAJ, respectively, according to the previous results in Table 5. We see that, as the number of segments increases, all four models yield decreasing hit ratio and recall. This is because all these models are based on recurrent neural networks, the performance of which are known to be negatively impacted by the length of the input sequence. Similar to the observations from Table 5, SARN outperforms the self-supervised competitor SRN2Vec consistently in all three metrics across all length settings. SARN* further improves over SARN. It outperforms the supervised competitor NEUTRAJ in most cases except in HR@5 when the maximum number of segments is 60 and in HR@20 when the maximum number of segments is 180. This demonstrates the robustness of the embeddings learned by our models, which enables them to be used for embedding long trajectories.

5.2.3 *Shortest-path Distance Prediction.*

Setup. The third task considers encoding the relative position of the road segments, i.e., to predict the *shortest-path distance* (SPD) [16, 28] between (the mid-point of) two road segments.

We consider directed graphs, which is different from the setting in the state-of-the-art model for the task (i.e., RNE [16], which uses undirected graphs). We thus cannot follow RNE and use the L_1 distance between two road segment embeddings directly as their predicted SPD. Instead, for each model (including RNE), we train an FFN with a single hidden layer of 20 nodes (using the MSE loss) to predict the SPD of two road segments, given the difference in each dimension between the corresponding embeddings produced by the model. We follow RNE and randomly sample 1‰ reachable origin-destination pairs from all road segment pairs for training and 0.01‰ pairs for testing. We measure the model performance by the *mean absolute error* (MAE) and *mean relative error* (MRE).

Results. Table 6 shows that SARN outperforms all other self-supervised baselines consistently as before. It reduces the prediction errors by up to 25.32% (in MAE on BJ) comparing with the best self-supervised baseline GCA. The spatial distance encoded in its embeddings help predict the relative positions of the road segments in the road network, since spatial distance and road network distance are often correlated.

SARN* further improves the performance and it outperforms RNE by up to 10.12% in MRE on SF (to be fair, RNE was designed for undirected graphs). We observe a performance gap between SARN* and the best supervised model HRNR. We emphasize that, unlike HRNR, SARN* is only fine-tuned but not trained from scratch for the task, as mentioned above. More importantly, spatial distance and road network distance do not always correlate (especially for road segments far away from each other), which can cause errors and negatively impact our models. In comparison, HRNR's three level hierarchical road network representation helps it learn the relative positions of the road segments better.

Here, we reiterate the strength of SARN and SARN* over the state-of-the-art self-supervised models, as our models are also self-supervised, while the supervised models may be limited by their applicability over different tasks.

5.2.4 *Road Networks with Different Sizes.*

Setup. We study the impact of the road network graph size on downstream task performance. We use SF as the baseline, since it has the largest number of road segments (i.e., 37,284). Specifically, we extract another two road networks in San Francisco, where the one with smaller area (**SF-S**) contains 19,540 road segments and the other with larger area (**SF-L**) has 74,016 road segments. The number of road segments in SF-S, SF and SF-L follows approximately two-fold increase. We have not run experiments on even larger road networks because one of the GCL baselines, GCA, already generates an out of memory error on SF-L. Similarly, we generate trajectory datasets within the regions of SF-S and SF-L respectively, following the aforementioned preprocessing of trajectories. For conciseness, we only report one of metrics for each downstream task.

Results. Table 8 shows the same trend as the previous experiments that is SARN consistently outperforms other self-supervised methods, and SARN* outperforms the state-of-the-art supervised methods on road property prediction and trajectory similarity prediction. Next, we discuss the results respectively.

Table 8: Downstream Task Results on Road Networks of Different Sizes

Category	Method	Road Property Predication F1 (%)			Trajectory Similarity Prediction HR@5 (%)			Shortest-Path Distance Predication MRE (%) (Smaller values are better)		
		SF-S	SF	SF-L	SF-S	SF	SF-L	SF-S	SF	SF-L
Self-supervised	node2vec	60.48±1.23	60.99±1.55	55.56±2.27	34.24±1.28	25.42±0.91	18.03±3.50	60.24±0.49	59.59±3.64	62.38±0.85
	SRN2Vec	56.00±2.06	54.85±0.51	43.27±1.63	62.43±1.13	59.73±0.94	54.71±2.04	59.38±0.35	62.94±0.32	64.56±0.22
	GraphCL	89.61±1.21	87.82±2.28	82.46±2.75	55.40±2.60	42.42±2.65	36.96±2.83	10.13±0.35	12.31±1.84	22.66±3.16
	GCA	88.20±0.87	87.89±1.91	OOM	58.89±1.11	52.48±2.36	OOM	10.06±0.67	8.76±0.47	OOM
	SARN	**92.44±0.95**	**91.21±0.71**	**86.62±1.00**	**68.25±1.60**	**68.80±0.92**	**68.73±1.57**	**8.75±0.13**	**7.53±0.69**	**7.45±0.15**
	Gain (%)	+3.16	+3.78	+5.04	+9.32	+15.19	+25.63	+13.02	+14.04	+67.12
Fine-tuned	**SARN***	**93.29±0.84**	**92.68±1.36**	**90.91±0.72**	**74.02±1.31**	**75.67±0.40**	**74.54±2.46**	**8.15±0.15**	**7.21±0.28**	**6.38±0.11**
Supervised	HRNR	72.79±2.38	89.42±0.86	OOM	67.14±6.20	65.50±0.89	OOM	5.58±0.12	4.85±0.34	OOM
	NEUTRAJ	-	-	-	68.81±2.27	72.94±2.40	69.29±4.27	-	-	-
	RNE	77.12±3.02	87.71±0.82	80.15±3.45	73.23±0.63	69.59±3.73	65.75±1.43	8.75±0.26	7.94±0.13	7.79±0.12
	Gain (%)	+20.97	+3.65	+13.42	+1.08	+3.74	+7.58	-46.06	-48.65	+18.10

On road property prediction, as the road network size increases, the F1 score of all self-supervised methods is decreased except that of node2vec on SF. SARN keeps superior than other self-supervised baseline methods by achieving up to 5.04% improvement on the largest SF-L dataset. After fine-tuning, SARN* outperforms other supervised methods as before, where the improvement is up to 20.97% on SF-S. Note that, GCA and HRNR are out of memory during the training process on SF-L. This is because GCA uses all vertices as negative samples in each iteration, and HRNR stores several different adjacency matrices and corresponding mapping relation. Both will take more GPU memory than other methods.

On trajectory similarity prediction, SARN and SARN* are persistently superior than other self-supervised methods and supervised methods with the performance gains by up to 25.63% and 7.58%, respectively. More importantly, SARN and SARN* are less affected by the road network size than other baseline methods except NEUTRAJ which also shows such strength. This is mainly due to the proposed spatial distance-based negative sampling strategy which can provide comprehensive negative samples from both local and global perspective even on a large road network.

On shortest-path distance prediction, SARN keeps achieving better performance (i.e., lower MREs) than other self-supervised methods, and SARN* further improves the performance comparing with SARN. We also notice that our methods SARN and SARN* have a better scalability than others especially GCA and HRNR which are the best self-supervised and supervised baseline methods on small road networks respectively. As GCA and HRNR are not applicable on large road networks, the improvement gains of SARN and SARN* rise to 67.12% and 18.10% on SF-L respectively.

5.3 Embedding Learning Efficiency

We report the embedding learning time of SARN and compare it with those of the other self-supervised models. We omit the learning times of the supervised models as they depend on the prediction models used for the different tasks, which are not our focus. We also omit the learning time of SARN*. Its initial embedding learning time is the same as that of SARN, while its fine-tuning is part of the prediction model training process, the time of which is task dependant.

From Fig.4, we can see that SARN is consistently and substantially faster than the best GCL based competitor GCA (in terms of prediction accuracy). The advantage is up to 5.59 times (on SF). This is because, as discussed in the cost analysis, GCA uses all road segments from both graph views as negative samples, which incurs high computation costs. SRN2Vec and GraphCL are the fastest, because SRN2Vec trains a simple FFN, and GraphCL does not maintain queues for the negative samples between mini-batches. Although SARN is slower than these two models, it still learns the embeddings of much higher quality in less than an hour, as evidenced by the high accuracy across different datasets and tasks shown above.

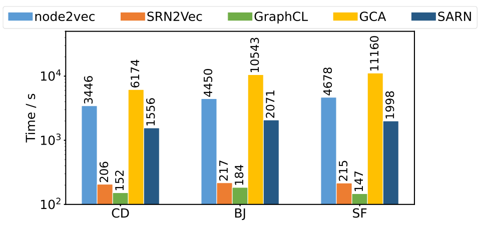


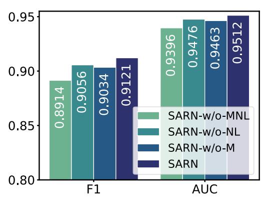
Figure 4: Embedding learning times

5.4 Ablation Study

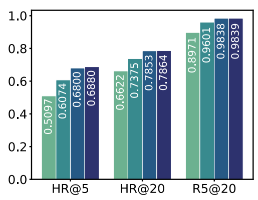
We study the effectiveness of the proposed components of SARN on SF for conciseness, since the results on other two datasets show the similar pattern. We use the following model variants:

- **SARN-w/o-MNL** is SARN without the spatial similarity matrix, the spatial distance-based negative sampling strategy, and the two-level loss function. It uses the InfoNCE loss (Eq. 2) with weighted topological edge-based graph augmentation and random negative sampling.
- **SARN-w/o-NL** is SARN without the spatial distance-based negative sampling strategy and the two-level loss function, i.e., it uses the spatial similarity matrix in graph encoding and our proposed graph augmentation strategy.
- **SARN-w/o-M** is SARN without spatial similarity matrix, i.e., it uses the weighted topological edge-based graph augmentation, the spatial distance-based negative sampling, and the two-level loss function.

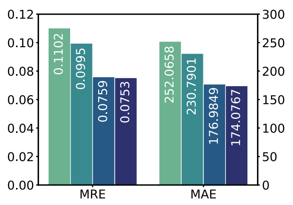
All sub-figures in Fig. 5 show that the model performs better as more proposed components are incorporated. This confirms the effectiveness of the components and the importance of spatial knowledge in road network embedding. For example, on trajectory similarity prediction (Fig. 5b), SARN-w/o-NL and SARN-w/o-M improve over SARN-w/o-MNL by 19.18% and 33.41% in HR@5, while SARN which has all proposed components further improves by 1.18% over SARN-w/o-M. Considering the difference of the proposed components, SARN-w/o-M outperforms SARN-w/o-NL on shortest-path distance prediction



(a) Road property prediction

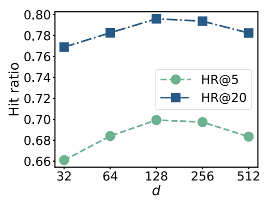


(b) Traj. similarity prediction

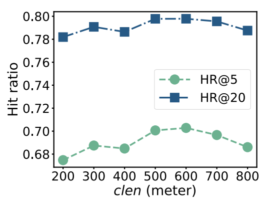


(c) Shortest-path distance prediction

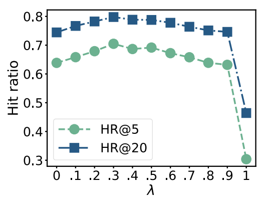
Figure 5: Ablation study results (Smaller values are better in Fig. 5c.)



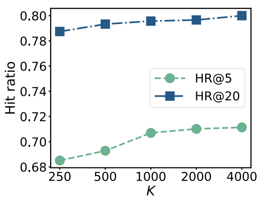
(a) Varying *d*



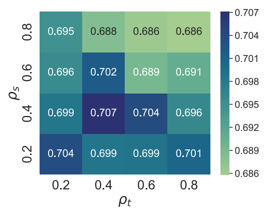
(b) Varying *clen*



(c) Varying λ



(d) Varying K



(e) Varying ρ_t and ρ_s

Figure 6: Impact of parameters on trajectory similarity prediction

(Fig. 5c), while SARN-w/o-NL outperforms SARN-w/o-M on road property prediction (Fig. 5a). Nevertheless, SARN retains the best performance, which shows that the spatial similarity matrix and the spatial distance-based negative sampling strategy (and the two-level loss) compensate for the errors made by each other.

5.5 Parameters Study

We study the impact of four parameters in SARN: the embedding dimensionality d, the cell side length *clen*, the loss trade-off λ, and the edge corruption rates ρ_t and ρ_s. We present the results on the trajectory similarity prediction task over SF in Fig. 6. We omit the results on the other tasks and datasets since the performance patterns are similar.

Impact of the embedding dimensionality d. Fig. 6a shows the hit ratios of SARN as d varies from 32 to 512. The hit ratios increase with d at start, where a larger d helps encode more information. The hit ratios reach the peak at $d = 128$ and decrease afterwards as d increases further. This is because now the embedding size has become too large, which triggers the over-fitting problem and impacts the test performance.

Impact of the cell side length *clen*. Fig. 6b shows the hit ratios of SARN as *clen* increases from 200 to 800 meters. We fix the total number of negative samples to 1,000 via adjusting the size of each queue as the number of cells changes with *clen*. The hit ratios also first increase with *clen*. Considering that the road segments are 70 meters in length on average (cf. Table 3), a 200×200 cell may not provide sufficient local negative samples. The best results are achieved when *clen* = 600, and the hit ratios decrease as *clen* increases further. This is because now there are too many local negative samples (and insufficient global negative samples) to be learned from, which may contribute more noise and confuse the model.

Impact of the loss trade-off parameter λ. Fig. 6c shows the hit ratios as λ increases from 0 to 1. Parameter λ controls the weight of the two loss terms in our loss function (Eq. 17). Specially, the loss function degenerates to a global contrastive loss (Eq. 16) when $\lambda = 0$, and it becomes to a local contrastive loss (Eq. 15) when $\lambda = 1$. The hit ratios first increase and then decrease as λ increases. The hit ratios are the highest when $\lambda \in [0.3, 0.5]$. This suggests that both our global and local loss terms play an important role in the learning process. A skewed loss function towards either the global loss or the local loss will cause the hit ratios to deteriorate. In particular, when $\lambda = 1$, the hit ratios drop significantly. This is because all negative samples come from the local negative samples, which severely limits the diversity of negative samples and loses the perspective of the whole embedding space.

Impact of the total size of negative sample queues K. Fig. 6d shows the hit ratios as K varies from 250 to 4,000. Parameter K controls the number of negative samples used in each training iteration. The results show a similar trend observed in Ref. [15], i.e., SARN benefits from a larger K in prediction accuracy. However, a larger K also leads to a higher training time for each training epoch. To balance the model effectiveness and efficiency, we use 1,000 as the default K value in the experiments, since there is a significant increase in HR@5 from $K = 500$ to 1,000.

Impact of the edge corruption rates ρ_t and ρ_s. Fig. 6e shows the HR@5 values as ρ_t and ρ_s vary from 0.2 to 0.8. The best performance is observed when both corruption rates are 0.4. We also see that when both corruption rates are greater than 0.5, HR@5 becomes worse than that when both corruption rates are less than 0.5. This is expected, as high corruption rates lead to sparse graphs, which suffer from insufficient neighboring vertices for contextual information learning. The results also show the importance of the spatial edges – corrupting the spatial edges (i.e., larger ρ_s) causes the hit ratios to drop faster. This is evidenced by that the hit ratios in the upper left triangle are lower (i.e., having lighter colors) than those of the lower right triangle. For example, the hit ratio at $\rho_s = 0.8$ and $\rho_t = 0.2$ is 0.695, which is lower than that at $\rho_s = 0.2$ and $\rho_t = 0.8$, which is 0.701.

6 CONCLUSIONS

We propose a self-supervised road network embedding model named SARN based on graph contrastive learning. Unlike existing models that learn dedicated embeddings for specific applications, SARN learns generic road network embeddings that can be applied to a wide range of downstream applications. We propose a spatial similarity matrix, a spatial importance-based graph augmentation strategy, a spatial distance-based negative sampling strategy, and a two-level contrastive loss function to guide SARN to learn both topology and spatial structure of road networks. Experiments on three downstream tasks over three real road networks show that SARN outperforms state-of-the-art self-supervised models consistently by up to 34% (in hit ratio). SARN also has comparable performance to supervised models, and even outperforms them after fine-tuning on two of the downstream tasks.

REFERENCES

[1] 2022. DiDi GAIA Open Dataset. https://outreach.didichuxing.com/research/opendata/.

[2] 2022. Haversine distance. https://en.wikipedia.org/wiki/Haversine_formula.

[3] 2022. OpenStreetMap. https://www.openstreetmap.org/.

[4] 2022. OpenStreetMap Road Types. https://wiki.openstreetmap.org/wiki/Key:highway.

[5] Helmut Alt and Michael Godau. 1995. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications* 5, 01n02 (1995), 75–91.

[6] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. 2020. Unsupervised Learning of Visual Features by Contrasting Cluster Assignments. In *NeurIPS*. 9912–9924.

[7] Yanchuan Chang, Jianzhong Qi, Egemen Tanin, Xingjun Ma, and Hanan Samet. 2021. Sub-trajectory Similarity Join with Obfuscation. In *SSDBM*. 181–192.

[8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *ICML*. 1597–1607.

[9] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. 2020. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297* (2020).

[10] David H. Douglas and Thomas K. Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization* 10, 2 (1973), 112–122.

[11] Tao-Yang Fu and Wang-Chien Lee. 2020. TremBR: Exploring road networks for trajectory representation learning. *ACM Transactions on Intelligent Systems and Technology* 11, 1 (2020), 1–25.

[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. 855–864.

[13] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1025–1035.

[14] Kaveh Hassani and Amir Hosein Khasahmadi. 2020. Contrastive multi-view representation learning on graphs. In *ICML*. 4116–4126.

[15] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *CVPR*. 9729–9738.

[16] Shuai Huang, Yong Wang, Tianyu Zhao, and Guoliang Li. 2021. A Learning-based Method for Computing Shortest Path Distances on Road Networks. In *ICDE*. 360–371.

[17] Tobias Skovgaard Jepsen, Christian S. Jensen, and Thomas Dyhre Nielsen. 2019. Graph convolutional networks for road networks. In *SIGSPATIAL*. 460–463.

[18] Eamonn Keogh and Chotirat Ann Ratanamahatana. 2005. Exact indexing of dynamic time warping. *Knowledge and Information Systems* 7, 3 (2005), 358–386.

[19] Ruiyuan Li, Huajun He, Rubin Wang, Sijie Ruan, Yuan Sui, Jie Bao, and Yu Zheng. 2020. TrajMesa: A distributed NoSQL storage engine for big trajectory data. In *ICDE*. 2002–2005.

[20] Xiucheng Li, Gao Cong, Aixin Sun, and Yun Cheng. 2019. Learning travel time distributions with deep generative model. In *WWW*. 1017–1027.

[21] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S. Jensen, and Wei Wei. 2018. Deep representation learning for trajectory similarity computation. In *ICDE*. 617–628.

[22] Yan Lin, Huaiyu Wan, Shengnan Guo, and Youfang Lin. 2021. Pre-training Context and Time Aware Location Embeddings from Spatial-Temporal Trajectories for User Next Location Prediction. In *AAAI*. 4241–4248.

[23] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. 2009. Map-matching for low-sampling-rate GPS trajectories. In *SIGSPATIAL*. 352–361.

[24] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *ICLR Workshop*.

[25] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

[26] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online learning of social representations. In *KDD*. 701–710.

[27] Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. 2009. A parsimonious model of mobile partitioned networks with clustering. In *International Communication Systems and Networks and Workshops*. 1–10.

[28] Jianzhong Qi, Wei Wang, Rui Zhang, and Zhuowei Zhao. 2020. A Learning Based Approach to Predict Shortest-Path Distances. In *EDBT*. 367–370.

[29] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. GCC: Graph contrastive coding for graph neural network pre-training. In *KDD*. 1150–1160.

[30] Huimin Ren, Sijie Ruan, Yanhua Li, Jie Bao, Chuishi Meng, Ruiyuan Li, and Yu Zheng. 2021. MTrajRec: Map-Constrained Trajectory Recovery via Seq2Seq Multi-task Learning. In *KDD*. 1410–1419.

[31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[32] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2018. Deep Graph Infomax. In *ICLR*.

[33] Jingyuan Wang, Ning Wu, Wayne Xin Zhao, Fanzhang Peng, and Xin Lin. 2019. Empowering A* search algorithms with neural networks for personalized route recommendation. In *KDD*. 539–547.

[34] Meng-Xiang Wang, Wang-Chien Lee, Tao-Yang Fu, and Ge Yu. 2019. Learning embeddings of intersections on road networks. In *SIGSPATIAL*. 309–318.

[35] Meng-Xiang Wang, Wang-Chien Lee, Tao-Yang Fu, and Ge Yu. 2020. On Representation Learning for Road Networks. *ACM Transactions on Intelligent Systems and Technology* 12, 1 (2020), 1–27.

[36] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Timos Sellis, and Xiaolin Qin. 2019. Fast large-scale trajectory clustering. *PVLDB* 13, 1 (2019), 29–42.

[37] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Zizhe Xie, Qizhi Liu, and Xiaolin Qin. 2018. Torch: A search engine for trajectory data. In *SIGIR*. 535–544.

[38] Tongzhou Wang and Phillip Isola. 2020. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *ICML*. 9929–9939.

[39] Zheng Wang, Cheng Long, Gao Cong, and Ce Ju. 2019. Effective and efficient sports play retrieval with deep representation learning. In *KDD*. 499–509.

[40] Ning Wu, Xin Wayne Zhao, Jingyuan Wang, and Dayan Pan. 2020. Learning Effective Road Network Representation with Hierarchical Graph Neural Networks. In *KDD*. 6–14.

[41] Di Yao, Gao Cong, Chao Zhang, and Jingping Bi. 2019. Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach. In *ICDE*. 1358–1369.

[42] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *NeurIPS* (2020), 5812–5823.

[43] Haitao Yuan and Guoliang Li. 2019. Distributed in-memory trajectory similarity search and join on road network. In *ICDE*. 1262–1273.

[44] Haitao Yuan, Guoliang Li, Zhifeng Bao, and Ling Feng. 2020. Effective travel time estimation: When historical trajectories over road networks matter. In *SIGMOD*. 2135–2149.

[45] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. 2011. T-drive: Enhancing driving directions with taxi drivers' intelligence. *TKDE* 25, 1 (2011), 220–232.

[46] Rui Zhang, Yacheng Rong, Zilong Wu, and Yifan Zhuo. 2020. Trajectory Similarity Assessment On Road Networks Via Embedding Learning. In *IEEE International Conference on Multimedia Big Data*. 1–8.

[47] Yanqiao Zhu, Yichen Xu, Qiang Liu, and Shu Wu. 2021. An Empirical Study of Graph Contrastive Learning. In *NeurIPS (Datasets and Benchmarks Track)*.

[48] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2020. Deep Graph Contrastive Representation Learning. In *ICML Workshop on Graph Representation Learning and Beyond*.

[49] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph contrastive learning with adaptive augmentation. In *WWW*. 2069–2080.