Learning Region Similarities via Graph-based Deep Metric Learning

Yunxiang Zhao, Jianzhong Qi, Bayu D. Trisedya, Yixin Su, Rui Zhang, Hongguang Ren

Abstract—Region similarity learning plays an essential role in applications such as business site selection, region recommendation, and urban planning. Earlier studies mainly represent regions as bags of points of interest (POIs) for region similarity comparisons, which cannot fully exploit the spatial features of the regions. Recently, researchers propose to use deep neural networks to exploit spatial features such as POI geo-coordinates and categories, which have produced more accurate and robust region similarity learning results. However, many useful features such as the height and size of a POI, and the distance and relative importance between the POIs, are still overlooked in these methods. To take advantage of such features, we propose to represent regions as graphs, where nodes are POIs with rich features such as height, size, and hexagonal coordinates, while edges are the relationships between POIs formulated by their road network distances. To capture POIs' importance, we weigh them by their height and size. Since there is limited availability of ground-truth region similarity data, we propose a contrastive learning-based multi-relational graph neural network (C-MPGCN) for region similarity learning based on the graph representations. To generate data for model training, we propose a soft graph edit distance (SGED) based algorithm to generate triples of similar and dissimilar graphs of a given graph (representing a given region) based on the POI weights. Experimental results show that C-MPGCN outperforms the state-of-the-art methods for region similarity learning consistently with an improvement of at least 8.6% and 9.4% in terms of MRR and HR@1, respectively.

Index Terms—Spatial Data Analysis; Region Similarity Learning; Graph Convolutional Network, Hexagonal Representation.

1 Introduction

Region similarity learning provides a way to transfer knowledge from known regions to new (or unknown) regions, which can facilitate applications such as business site selection [1], point of interest (POI)/region recommendation [2], [3], and urban planning [4], [5]. For example, when the owner of a popular restaurant plans to expand their business, region similarity learning can help identify candidate regions for a new branch. This is done based on the similarities between the regions of interest and the region within which the current restaurant locates. Intuitively, similar regions may have similar business opportunities. This reduces the number of regions to be considered, and thus it facilitates the decision-making process.

Existing methods for region similarity learning are mainly based on comparing the bags of POIs (e.g., POI categories) that represent the input regions [6], [7]. These methods have ignored the spatial relationships among the POIs. For example, each of the two regions in Figure 1(a) has a set of four POIs. Both sets have POIs from the same four categories: health service, education, shopping, and groceries. By using bags of POI categories, the two regions will be considered the same [8]. However, the two regions are actually different. Compared with the lower region in Figure 1(a), the POIs in the upper one are larger in size (i.e., a hospital is usually much larger than a clinic) and are spatially less distant from each other.

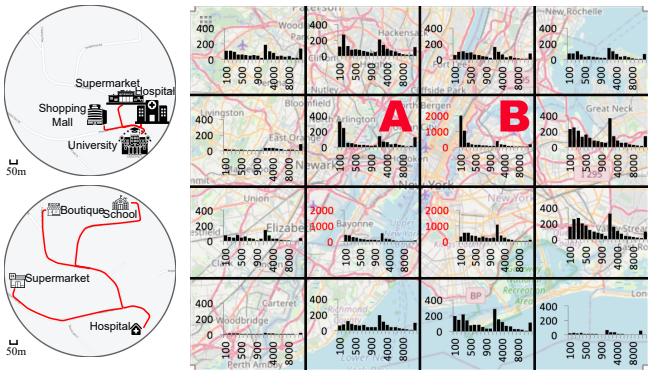


Fig. 1. (a) Two regions with POIs of the same categories; (b) Histograms of POI sizes (in square meters, m^2) across New York City.

Recently, Liu et al. [1] propose a deep metric learning method that considers POI geo-coordinates in addition to POI categories, while Jin et al. [9] further capture the hierarchical relationships between POI categories via convolutional neural networks (CNNs). For example, a Japanese restaurant and a Korean restaurant are considered the same at a coarse-grained level that takes all dining places as restaurants, while they are different at a fine-grained level that differentiates dining places by their cuisines. However, POI features such as height, size, and road network distance relationships among POIs have not been considered. We summarize the limitations of existing region similarity

- *Yunxiang Zhao and Hongguang Ren are with Beijing Institute of Biotechnology, China. E-mail: zhaoyx1993@163.com; bioren@163.com*
- *Jianzhong Qi and Yixin Su are with The University of Melbourne, Australia. E-mail: jianzhong.qi@unimelb.edu.au; yixins1@student.unimelb.edu.au*
- *Bayu D. Trisedya is with SEEK, Australia. E-mail: bayu.trisedya@rmit.edu.au*
- *Rui Zhang is with www.ruizhang.info, China. E-mail: rayteam@yeah.net*

**Jianzhong Qi and Hongguang Ren are the corresponding authors.*

learning methods as follows:

- Existing methods overlook many useful POI features such as height and size for region similarity learning. For example, in Figure 1(b), we show histograms of POI of different sizes (in m^2) in different regions across New York City. We can see that similar POI size distribution is a strong indicator of similar regions, e.g., regions A and B in the figure are similar. They are both in urban areas, which have many POIs with a small size (e.g., small shops).
- Existing deep learning-based region similarity learning methods partition the map with a square-shaped grid of a manually defined granularity. The POIs are encoded by the IDs of the cells in which the POIs lie. Neighboring POIs of similar Euclidean distances to a POI p may have different cell ID offsets to that of p, due to the square grid-based space partitioning (detailed in Section 3.2). This may lead to a discriminative effect on the neighboring POIs, especially for site selection applications. Moreover, the non-Euclidean relationships such as the relative importance and road network distance between POIs are important, which cannot be fully captured either.

In this paper, we advance region similarity learning by further exploiting the following POI features: (i) The non-Euclidean relationships between POIs – we propose to use a graph to represent a given region, where the POIs are the graph nodes, and the edge weights are defined based on the road network distance between the POIs. (ii) The POI importance – we embed rich POI features that reflect the POI importance such as height and size via an entropy-based algorithm. The embedded features are used as node features for the graph (region) embedding learning.

To learn the region representations, we propose a multi-relational graph convolutional network (MPGCN) model. MPGCN differentiates neighboring nodes of a target node in different road network distance ranges during the aggregation process, such that POIs in different road network distance ranges contribute differently to the embedding of a target POI. MPGCN further follows the DiffPool model [10] and applies the pooling technique to learn representative nodes from each GCN layer to be passed onto the next layer, thus reflecting the relative POI importance in region representation.

Building upon MPGCN, we propose a triplet contrastive learning-based neural network (C-MPGCN) for region similarity learning. To train C-MPGCN, we need triples formed by pairs of similar regions and dissimilar regions. Due to the unavailability of such ground-truth data, we generate training data by editing known regions. Given a region r to be fed into C-MGGCN for similarity learning, we generate a region similar to r and another region dissimilar to r. We propose a graph edit distance-based measure named soft graph edit distance (SGED) to apply "edits" on (the POIs of) r. Different from the original graph edit distance (GED), which measures the similarity between graphs, we use SGED to guide the generation of similar and dissimilar graphs. The SGED-based algorithm gives non-uniform edit costs according to the importance and uniqueness of the POIs related to an edit operation. Intuitively, a unique POI in a region with a large size or height is more important in learning the representation of a region, and editing such a POI would make the resultant region less similar to the original region than editing other less unique POIs. Moreover, our SGED-based algorithm computes a fuzzy similarity score rather than an exact one like GED, which helps reduce the time complexity to generate our model training data. To summarize, we make the following contributions:

- We are the first to use graph representation for region similarity learning, where nodes are rich POI features and edges represent road network distances between POIs. We propose a variant of GCN named MPGCN to learn region representations, which captures both POI features and POI explicit/implicit relationships.
- We extend MGGCN with a triplet contrastive learning-based model, named C-MPGCN, to learn to predict region similarity. To generate training data for C-MPGCN, we define SGED (an extension of GED) and propose an SGED-based algorithm to generate similar and dissimilar regions for a given region.
- We collect a dataset from New York City with 30,832 POIs, each of which has its category, geo-coordinates, size, and height as the features. We will release this dataset for public use. We run extensive experiments on this dataset. The results show that C-MPGCN improves the accuracy of similar region prediction consistently with an improvement of at least 8.6% and 9.4% in terms of MRR and HR@1, respectively.

We organize the rest of this paper as follows. We review related work in Section 2 and detail the proposed C-MPGCN model in Section 3. We report experimental results in Section 4 and conclude the paper in Section 5.

2 Related Work

We review existing studies on region similarity learning and graph similarity learning, which are the foundations of our proposed model.

2.1 Region Similarity Learning

In region similarity learning, the inputs are regions cropped from a given map, and the aim is to compute the similarity between two regions. Most existing works represent a region by aggregated attributes of the region, e.g., the number of POIs (or POI categories) in the region. For example, Le et al. [11] find that the earth mover's distance between vector representations of different regions is an effective measure to compute similar regions. Shen et al. [12] measure the region similarity by comparing the spatial feature vectors that consist of both the POIs' categories and their geo-coordinates from the two regions. Liu et al. [8] further incorporate the diversity and distribution of POIs when learning the region similarities. Wang et al. [6] use POI features (e.g., category and reviews) and taxi flow information for crime predictions in different regions.

Recently, deep learning has been used for region similarity learning. For example, Liu et al. [1] propose a CNN model via a triplet-based [13] learning schema on the POI category and geo-coordinates. Jin et al. [9] further consider the hierarchical relationships of POI categories. There are

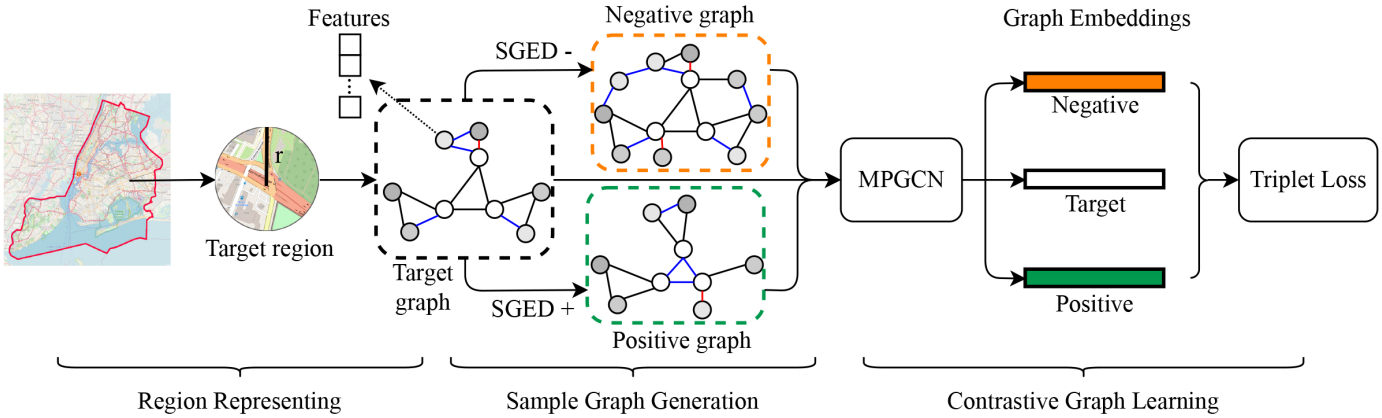


Fig. 2. Structure of C-MPGCN (best view in color).

also works that learn region embeddings for downstream tasks such as traffic accident risk forecasting and economic growth prediction [14], [15], [16], where region features are captured in a coarse-grained manner – each region is represented as a graph node (instead of a graph as done in our work), and each edge represents the relationship between two different regions. These methods have not exploited features such as POI height and size, which are also important in region similarity comparison. Moreover, they have not captured non-Euclidean relationships such as road network distances and relative POI importance within a region.

2.2 Graph Similarity Learning

To represent a region, we observe that not only the POIs in a region but also their connectivities are important. The network of POIs in a region motivates us to use graph similarity learning for region similarity learning. Graph edit distance (GED) [17] and maximum common subgraph (MCS) [18] are widely used for measuring the similarities between two graphs, because they are domain-agnostic [19]. In this paper, we focus on GED as MCS is a special case of GED under a particular cost function [20], and MCS only considers the structural changes of the graphs. Computing the exact GED is NP-complete [21], and the state-of-the-art algorithms cannot compute the exact GED within a reasonable time for graphs with more than 16 nodes [22].

To tackle the computation complexity issue, existing studies use pruning strategies [23], [24] or produce approximate GED in a fast and heuristic way [23], [25], [26]. These algorithms require complex design and implementation based on discrete optimization or combinatorial search. Their time complexities are still polynomial or even sub-exponential to the number of nodes in graphs [27], [28]. For example, Bougleux et al. [25] treat GED as a quadratic assignment problem (QAP) of nodes in different graphs [29]. They adapt the integer projected fixed-point algorithm originally designed for QAP to compute an approximate GED by finding a local minimum.

Recently, researchers transform graph similarity computation into a learning problem [27], [30]. Bai et al. [27] formulate graph similarity learning as a regression task, where a GCN and attention layers are trained by precomputed GED scores. Li et al. [31] use graph neural networks to learn graph embeddings and use a Graph Matching Network (GMN) to compute graph similarity through cross-graph attention-based matching. Wang et al. [32] map each graph to an embedding vector independently, and they compute graph similarity in the vector space. This enables precomputing and indexing of the graph embeddings and hence efficient retrieval of similar graphs using fast nearest neighbor search data structures such as k-d trees [33]. However, the model training of methods above requires a large volume of labeled data which is difficult to obtain.

3 Proposed Model

We first present an overview of C-MPGCN in Section 3.1, followed by the region representation for similarity learning in Section 3.2. We present the definition of SGED and the SGED-based algorithm to generate similar and dissimilar region triples for training C-MPGCN, and the details on POI weighting in Section 3.3. We present the MPGCN module to capture POIs' road network distances and relative importance in Section 3.4.

3.1 Model Overview

Given a query region q and an area of interest D from a map, we aim to return the most similar region $r \in D$ of q. For simplicity and to avoid the impact of region shape and size, both q and r have the same circular shape and are of the same radius γ. Such a region shape setting suits the motivating application for business site selection, where a region of a certain radius centered at a candidate site is of interest when selecting a site for a new business. Note that our model can be used to learn region similarity for regions of arbitrary shape when such regions are available, e.g., from algorithms that partitions the map into irregular regions [34], [35], [36].

Since "region similarity" is a somewhat subjective concept, and there is no universal definition for it, we take a fuzzy approach and apply triplet contrastive learning techniques to train a model named C-MPGCN to predict region similarity. The basic idea of triplet contrastive learning is that, given a data sample of interest t, we fetch (or

generate) a similar data sample (i.e., positive sample t_+ and a dissimilar data sample (i.e., negative sample t_-). The triple of data samples $\langle t, t_+, t_- \rangle$ is fed into a neural network F for training. A conditional function $||F(t) - F(t_+)||_2^2 + \alpha < ||F(t) - F(t_-)||_2^2$ is used for model training, which guides F to yield embeddings between t and t_+ that are closer (i.e., with a shorter distance in the embedding space) than the embeddings between t and t_- by a margin of α (a hyperparameter). Based on the conditional condition, the loss function we use when training in a batch mode is:

$$\mathcal{L} = \sum_{i}^{b} \max(||F(t_i) - F(t_{i+})||_2^2 + \alpha - ||F(t) - F(t_{i-})||_2^2, 0) \quad (1)$$

where b denotes the number of triples in a batch.

In our triplet contrastive learning-based model C-MPGCN, as shown in Figure 2, a region r is a data sample of interest. For model training, given the map data of a certain area, we randomly select a POI p from the given area. The circular region centered at p with a radius of γ forms r. We use a POI as a region center to suit the motivating application of site selection, although our techniques apply directly if a random point is used as the region center instead. We extract all POIs in r to form a graph G_r of POIs to represent r, where each graph node is a POI (detained in Section 3.2) and the edges are defined based on road network distances between POIs (detailed in Section 3.4). We apply the proposed soft graph edit distance algorithms (detailed in Section 3.3) to edit G_r to generate a positive sample G_{r_+} and a negative sample G_{r_-}. These graphs then go through the proposed GCN variant MPGCN (detailed in Section 3.4) to be mapped into an embedding space. The embeddings produced by MPGCN are fed into the triplet loss function (Equations 1) to compute the model loss and obtain training signals for model parameter updates.

Once trained, given a query region q and an area of interest D from a map, we can use MPGCN to generate embeddings of q and any region in D. By comparing the embedding difference (e.g., Euclidean distance) between q and candidate regions in D, we can identify the region that is most similar to q. A geographical region is a 2-dimensional continuous space, which contains an infinite number of regions and is infeasible to search. Instead of using the brute-force method to enumerate all candidate regions in D, we take each POI in D as the origin and crop an area with a radius of γ to generate candidate regions, so as to improve the efficiency.

3.2 Region Representation

We represent a region r as a graph G_r where every node is a POI in r and every edge has a weight that represents the road network distance between two POIs. Since G_r will be fed into a GCN variant for embedding learning, the feature vector of each node and the weight of each edge in G_r play a critical role in the quality of the learned embedding. Next, we detail the node (POI) features and edge weights of G_r.

Node features. Our feature vector for each POI p has 37 dimensions, where 30 dimensions (a one-hot embedding) are used to represent the POI category obtained from OpenStreetMap [37], [38] (cf. Table 1), two dimensions are for POI longitude and latitude (integers), two dimensions are for POI height and size (floats) obtained from NYC Open Data[1], and the remaining are for POI hexagonal coordinates.

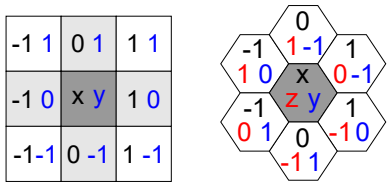


Fig. 3. (a) Cartesian coordinate system, (b) Cube coordinate system. The central cells are the cells that p lies in.

POI hexagonal coordinates. Existing deep learning-based region similarity learning methods partition the map into square-shaped grid cells with a manually defined granularity. The POIs are then encoded by the IDs of the cells that contain the POIs. This may discriminate neighboring POIs with similar Euclidean distances to a target POI, which is inferior for applications such as site selection. As shown in Figure 3a, suppose a target POI p lies in the center cell, and the cell IDs are the row and column numbers of the cells. POIs in the surrounding cells may have similar Euclidean distances to p. The cell IDs of the POIs in the four white cells will differ from those of p by 1 in both the row and the column numbers. In comparison, the cell IDs of the POIs in the four gray cells will differ from those of p by 1 in only the row or the column number.

Compared with square-shaped partitioning, a hexagonal partitioning of the space offers isotropic properties such that the nearest cells of a cell have the same distance to the cell regardless of the direction [39]. The Cube coordinate system (Figure 3b) is based on this partitioning scheme, where the coordinates of each cell are three-dimensional integers, i.e., $\langle h_x, h_y, h_z \rangle$. Cube is the most representative coordinate system [40] and retains the relative position information of different cells. We thus use such coordinates for the POIs in addition to the geo-coordinates, to enrich the location representation of the POIs.

Edge features. We formulate the adjacent matrix of a graph G as $\mathbf{A}$. Let $\mathbf{A}_{i,j}$ denote the connectivity between POIs p_i and p_j. We define $\mathbf{A}_{i,j}$ as:

$$\mathbf{A}_{i,j} = \begin{cases} 0, \ if \ dist(p_i, p_j) > \theta \\ \log_2 \ (\lceil dist(p_i, p_j)/dist_n \rceil) + 1, \ \text{otherwise} \end{cases} \quad (2)$$

where $dist(p_i, p_j)$ denotes the road network distance between p_i and p_j. We discretize the values of $dist(p_i, p_j)$ into a few categories where $dist_n$ is the span of a road network distance category (200 meters in our experiments) and "$\lceil \cdot \rceil$" is the ceiling function. This discretization allows us to learn different weights for neighbors in different road network distance categories. We consider POIs to be disconnected if their road network distance is over a threshold θ.

3.3 The Soft Graph Edit Distance Algorithm

Our *soft graph edit distance* (SGED) is based on graph edit distance (GED) [17], GED measures the "edit distance"

1. https://opendata.cityofnewyork.us/

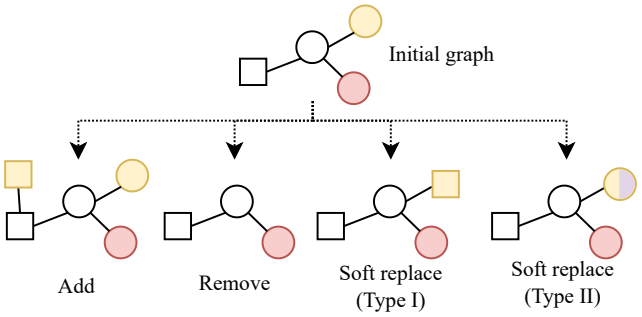


Fig. 4. SGED graph edit operations (best viewed in color; different shapes represent POIs of different categories, and different colors represent POIs with different feature values).

between two graphs by computing the number of "edits" required to convert one of the graphs to the other. There are three edit operations in GED, i.e., "remove", "replace", and "add" a node or an edge. Each operation contributes the same unit edit distance between two graphs. Applying these operations directly to our problem is suboptimal since different points of interest (POIs) contribute differently to the representation of a region as discussed earlier. Moreover, the original GED cannot be computed within a reasonable time for graphs with more than 16 nodes [22].

To address these limitations, we extend the classic notion of GED to account for the changes in node features while avoiding its high computation time complexity. This leads to our proposal of SGED. In this subsection, we first present the definitions of GED and SGED, as well as their differences. We then propose two algorithms that use SGED for generating positive and negative samples (i.e., similar and dissimilar regions of a target region for representation learning).

Definition 1. (**Graph Edit Distance**) Suppose that there are two graphs G_1 and G_2, the graph edit distance (GED) between G_1 and G_2 is defined as:

$$GED(G_1, G_2) = \min_{(e_1, e_2, \ldots, e_m) \in \mathbb{P}(G_1, G_2)} \sum_{i=1}^{m} \hat{\omega}_{e_i} \quad (3)$$

where $\mathbb{P}(G_1, G_2)$ denotes the set of all different sequences of edit operations that can transform G_1 into G_2, and $\hat{\omega}_{e_i} \geq 0$ is the cost of each graph edit operation e_i. Here, we focus on edit operations over the nodes, i.e., the set of graph edit operations considered in GED includes:

- Add: add a new node to the graph.
- Remove: remove an existing node from the graph.
- Replace: replace an existing node in the graph with a new node.

Definition 2. (**Soft Graph Edit Distance**) Suppose that there is a graph G_1 and another graph G_2 which is generated from G_1 via a sequence of randomly selected edits $e_1, e_2, \ldots, e_m$. The soft graph edit distance (SGED) between graph G_1 and G_2 is defined as:

$$SGED(G_1, G_2) = \sum_{i=1}^{m} \omega_{e_i} \quad (4)$$

where $\omega_{e_i} \geq 0$ is the cost of each graph edit operation e_i. Note that there may be another graph edit sequence that generates G_2 from G_1 and hence SGED-based graph edit sequence between G_1 and G_2 may not be unique. We allow this fuzziness in the definition of SGED-based graph edit distance for the following reason. The training of our region similarity learning model only requires graphs with an upper-bounded SGED-based graph edit distance to an initial graph G_1. To generate the graphs for model learning, we take a generative procedure by iteratively applying edit operations to graph G_1 until the costs of the edit operations reach a given SGED-based graph edit distance bound (Algorithms 1 and 2, detailed late). This generation procedure guarantees that the SGED-based graph edit distance between G_1 and any graph generated will not be greater than the given SGED-based graph edit distance bound. The actual SGED-based graph edit distance between G_1 and a graph generated may be smaller than the given bound. However, this does not impact our model learning, since our model learning target, i.e., region similarity, is a fuzzy concept. Like GED above, we focus on edit operations on the nodes. In particular, we consider the following graph edit operations in SGED (as illustrated by Figure 4):

- Add: add a new node to the graph.
- Remove: remove an existing node from the graph.
- Soft replace (Type I): replace an existing node in the graph with a new node.
- Soft replace (Type II): amplify the value of a feature of a node in the graph by a given ratio.

Comparing SGED with GED. SGED differs from GED in three aspects: (1) GED is defined based on the *minimum* editing cost to modify G_1 into G_2, while SGED simply measures the cost of a given sequence of graph edit operations, which is much more efficient to compute and enables more efficient model learning. (2) GED [17] typically gives the same (unit) cost to edits on different nodes, while SGED gives different costs to edits on different nodes (based on the importance of the POIs, cf. **POI weight** later). (3) GED does not edit the feature values of a node, while SGED has a "soft replace (Type II)" operation that replaces a POI by itself while modifying some feature value by a given ratio. This soft replace operation is critical, especially for generating a graph similar to a small graph with only a few POIs, where adding, removing, or replacing a POI may create a much different graph.

SGED for Positive and Negative Sample Generation. As summarized in Algorithm 1, given the graph representation G_r of a target region, the SGED-based algorithm generates a positive sample G_{r_+} as follows. We first initialize G_{r_+} to be the same as G_r (Line 1). We then generate edit operations randomly from the set of add, remove, and soft replace operations to the nodes in G_{r_+}. Graph G_{r+} is edited repeatedly, and the edit distances accumulated from the edits are recorded (Lines 2 to 22). Once the accumulated edit distance of G_{r_+} exceeds a threshold ϵ_+ with one more edit, the algorithm terminates and returns the resultant G_{r_+}. The generated G_{r_+} has less than ϵ_+ edit distance to G_r. Similarly, the SGED-based algorithm generates a negative sample G_{r_-} with more than ϵ_- edit distance to G_r to

Algorithm 1 SGED-based algorithm for positive sample generation

Input: A graph representation G_r of a target region.
Output: A positive sample G_{r_+} of G_r.

acc = 0;
ops = [];
$op = null$;
$G_{r_+} = G_r$;
$I = \sum_{i=1}^{n} s_{p_i} \cdot u_{p_i}$;
while $acc < \epsilon_+$ **do**
  // Do nothing for the first round;
  $G_{r_+} = G_{r_+}.update(op)$
  $op = randomOp()$;
  if op is add **then**
    $p' = randomPoi(All)$;
    $acc+ = (s_{p'} \cdot 1/(nc(G, p') + 1))/I$;
  else if op is $remove$ **then**
    $p = randomPoi(G_{r_+})$;
    $acc+ = (s_p \cdot u_p)/I$;
  else if op is $softReplace$ **then**
    $subOp = randomReplace()$;
    if $subOp$ is $newPoi$ **then**
      $p = randomPoi(G_{r_+})$;
      $p' = randomPoi(All)$;
      $acc+ = (s_p \cdot u_p + s'_p \cdot u'_p)/I$;
    else
      $p = randomPoi(G_{r_+})$;
      $acc+ = (s_p \cdot u_p \cdot w_j^f \cdot |\Delta|)/I$;
return G_{r_+}

Algorithm 2 SGED-based algorithm for negative sample generation

Input: A graph representation G_r of a target region.
Output: A negative sample G_{r_-} of G_r.

acc = 0;
ops = [];
$G_{r_-} = G_r$;
$I = \sum_{i=1}^{n} s_{p_i} \cdot u_{p_i}$;
while $acc < \epsilon_-$ **do**
  $op = randomOp()$;
  if op is add **then**
    $p' = randomPoi(All)$;
    $acc+ = (s_{p'} \cdot 1/(nc(G, p') + 1))/I$;
  else if op is $remove$ **then**
    $p = randomPoi(G_{r_-})$;
    $acc+ = (s_p \cdot u_p)/I$;
  else if op is $softReplace$ **then**
    $subOp = randomReplace()$;
    if $subOp$ is $newPoi$ **then**
      $p = randomPoi(G_{r_-})$;
      $p' = randomPoi(All)$;
      $acc+ = (s_p \cdot u_p + s'_p \cdot u'_p)/I$;
    else
      $p = randomPoi(G_{r_-})$;
      $acc+ = (s_p \cdot u_p \cdot w_j^f \cdot |\Delta|)/I$;
  $G_{r_-} = G_{r_-}.update(op)$;
return G_{r_-}

generate a negative sample G_{r_-}. Graph G_r, its positive sample G_{r_+} and its negative sample G_{r_-} form a resultant triple $\langle G_r, G_{r_+}, G_{r_-} \rangle$. Below, we define the cost of a POI edit, denoted by ω, assuming a graph G_r to be edited.

$$\omega = \begin{cases} s_{p'} \cdot 1/(nc(G_r, p') + 1), \textbf{add} \text{ a new POI } p' \\ s_p \cdot u_p, \textbf{remove} \text{ an existing POI } p \\ s_p \cdot u_p + s_{p'} \cdot u_{p'}, \textbf{soft replace (Type I)} \text{ an} \\ \qquad \text{existing POI } p \text{ by a new } p' \\ s_p \cdot u_p \cdot w_j^f \cdot |\Delta|, \textbf{soft replace (Type II)} \text{ the} \\ \qquad \text{feature value } j \text{ of POI } p \text{ by increasing} \\ \qquad \text{or decreasing with a ratio of } \Delta \end{cases} \quad (5)$$

where s_p ($s_{p'}$) and u_p ($u_{p'}$) denote the "importance" and "uniqueness" of POI p (p'), $nc(G, p')$ denotes the number of POIs in G_r that are in the same category as p', w_j^f denotes the weight of feature j. The uniqueness of u_p denotes how unique p is in terms of its category. Suppose $nc(G, p)$ denotes the number of POIs in G_r that are in the same category as p, then $u_p = 1/nc(G, p)$.

Intuitively, a POI that is more unique in G_r (i.e., with few other POIs in the same category of p) should introduce a larger edit distance when it is edited. For the add and soft replace (Type I) operations in Algorithm 1, the new POI p' is randomly selected from the map area used for model training. Further, for the add operation, we randomly select a position within the target region for the POI p'. For the remove and soft replace operations in Algorithm 1, the exiting POI p is randomly selected from G_{r+}. Further, for soft replace (Type II), we randomly select a feature j from POI p and update its value by a randomly selected ratio $\Delta \in (-1, 1)$.

Given a sequence of m SGED operations $e_1, ..., e_m$ with edit costs $\omega_{e_1}, \omega_{e_2}, \ldots, \omega_{e_m}$, their overall impact (i.e., total edit distance) to G_r is $\sum_{i=1}^{m} \omega_{e_i}$. We normalize this impact value by the importance and uniqueness values of all the POIs in G_r originally, i.e, $I = \sum_{i=1}^{n} s_{p_i} \cdot u_{p_i}$ where n is the number of POIs in G_r. The normalized SGED-based graph edit distance of m SGED operations, denoted by acc, is $acc = \sum_{i=1}^{m} \omega_{e_i}/I$.

This normalized acc is used with ϵ_+ and ϵ_- in Algorithm 1 and Algorithm 2 to control the generation of the positive and negative samples G_{r_+} and G_{r_-}. Below, we detail how the POI importance score s_p and the weight w_j^f of feature j are calculated.

POI weighting. We compute the important s_{p_i} of a POI p_i as a normalized aggregation of its non-coordindate feature values:

$$s_{p_i} = (\sum_{j=1}^{d} w_j^f \cdot p_i \cdot a_j) / \max_{p \in C(p_i)} \{\sum_{j=1}^{d} w_j^f \cdot p \cdot a_j\} \quad (6)$$

where $p_i \cdot a_j$ denotes the j-th non-coordinate feature value of p_i. In our implementation, we use two such features ($d = 2$), height and size, due to the limited data availability, although our technique generalizes to $d > 2$. $C(p_i)$ denotes the set of POIs that are in the same category of p_i. The weight of each feature, i.e, w_j^f, is used to reflect the relative importance of a feature dimension. Intuitively, a feature dimension carries a

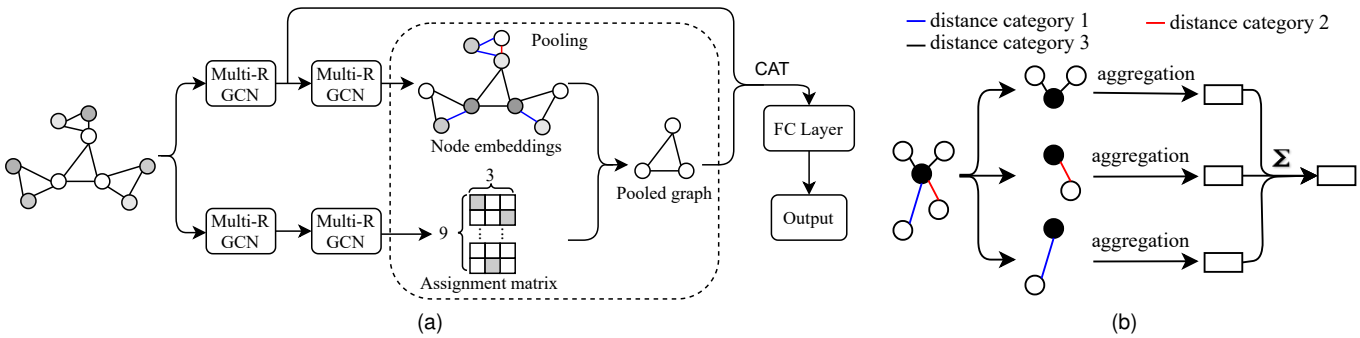


Fig. 5. (a) MPGCN Structure; (b) The Multi-R GCN block for a target node (the black solid one, best view in color).

heavier weight if the values of POIs in this dimension have a larger variance because such a feature dimension helps distinguish different POIs compared with other feature dimensions. We compute w_j^f for each POI category separately because the same feature dimension may contribute to the importance of POIs in different categories differently.

Given a set of n POIs from the entire training dataset, each with d-dimensional features, we compute w_j^f using a Shannon entropy-based ranking algorithm. Shannon entropy is a commonly used measurement of uncertainty in information theory [41]. The idea of entropy-based ranking is to compute the weights of different parameters (i.e., feature dimensions in our case) according to the parameter value distribution. If the values of a feature dimension vary greatly among the data samples (i.e., POIs), then the feature dimension is considered an important one, as the feature dimension brings more information based on the principle of Shannon entropy [42].

The input to the entropy-based ranking is an $n \times d$ decision matrix $\mathbf{M}$, where an element $\mathbf{M}_{ij}$ is the value of the i-th POI in the j-th feature dimension (e.g., a POI height). We normalize $\mathbf{M}$ with the min-max scaling as follows:

$$\mathbf{M}_{ij} = (\mathbf{M}_{ij} - \min(\mathbf{M}_{\cdot j})) \,/\, (\max(\mathbf{M}_{\cdot j}) - \min(\mathbf{M}_{\cdot j})) \quad (7)$$

This normalization is critical because different feature dimensions may have different scales (e.g., the maximum height of a POI is less than 1,000 meters, while the maximum size of a POI can be more than 50,000 square meters. After the normalization, the entropy of each feature dimension is computed as:

$$\mathbf{E}_j = -\ln(n)^{-1} \cdot \sum_{j=1}^{n} \mathbf{M}_{ij} \cdot \ln\,(\mathbf{M}_{ij}),\ \mathbf{M}_{ij} = \mathbf{M}_{ij} / \sum_{j=1}^{n} \mathbf{M}_{ij} \quad (8)$$

We then compute the weight of each feature dimension based on its entropy by:

$$w_j^f = \,(1 - \mathbf{E}_j)/(d - \sum_{j=1}^{n} \mathbf{E}_j),\ j = [1, d] \quad (9)$$

With the computed weight of each feature dimension, we generate positive and negative samples via the algorithms 1 and 2.

3.4 MPGCN

Given a graph G_r representing a region of interest or its generated positive/negative sample graphs, we pass it and its adjacency matrix $\mathbf{A}$ into a GCN variant named MPGCN that we propose for graph embedding. As shown in Figure 5a, MPGCN has two parallel branches. The upper branch uses two Multi-R GCN (cf. Figure 5b) blocks to learn the node embeddings. For each node i in the Multi-R GCN block, instead of taking all neighbors equally [10], [43], we perform message passing [44] from its neighbors with different relationships (i.e., different road network distance categories) separately, and we take the weighted sum as node i's new embedding $\mathbf{h}_i'$:

$$\mathbf{h}_i' = \sum_{\phi \in \Phi,\, g \in N_{i,\psi}} W_{\phi g} \mathbf{h}_g \quad (10)$$

where Φ denotes the set of different road network distance categories that node i's neighbors fall into. $N_{i,\phi}$ denotes the neighbors in category ϕ, W_ϕ denotes the weight of ϕ, $\mathbf{h}_g$ denotes the feature vector of node i's neighbor g.

The lower branch of MPGCN uses another two Multi-R GCN blocks to learn an assignment matrix for the input graph. The learned assignment matrix is used to select the nodes to be fed into the next model component, i.e., a fully connected (FC) layer. For example, the input graph in Figure 5a contains 9 nodes, and we keep only 3 of them for the FC layer. The learned assignment matrix in this example has three columns, and each column has only one element being 1 while all others are 0's. This lower branch learns to keep the more representative POIs from an input graph (i.e., to learn the relative importance among the POIs). After the pooling operation, we apply an FC layer on the remaining nodes to generate the final representation of an input graph.

4 Experiments

In this section, we evaluate C-MPGCN against state-of-the-art models and study the effectiveness of the model components.

4.1 POI categories

We collect the POIs from the OpenStreetMap dataset and study the POI type tags, which are shown as the subcategories in Table 1. The number next to each subcategory in the table denotes the number of POIs collected in that subcategory. To reduce the number of POI categories to be considered, we merged the subcategories into 30 categories, e.g., night club, gambling, and dance are merged

TABLE 1
POI Categories: the number next to each category denotes the number of POIs of that category.

Categories	Subcategories
#1	bridges (4)
#2	data_center (9)
#3	marketplace (9)
#4	cinema (19)
#5	construction (91)
#6	shelter (96), shed (1,713)
#7	nightclub (6), gambling (1), dance (1)
#8	ruins (1), tourism (1), mosque (1), monastery (1), tower (3), historical (4)
#9	waste_transfer_station (1), recycling (12)
#10	farm (1), farm_auxiliary (1), barn (8), stable (17), greenhouse (18)
#11	car_wash (26), vehicle_inspection (1), car_service (1), car_repair (1), car_sharing (2), bicycle_rental (4), car_rental (25)
#12	funeral_home (1), mortuary (1), crematorium (1), prison (19), grave_yard (31)
#13	bus_depot (1), transportation (3), bus_station (14), train_station (34)
#14	deckhouse (1), ship (1), submarine (1), boat_storage (1), boathouse (3), boat_rental (10), ferry_terminal (10), slipway (15), marina (60)
#15	motel (2), hotel (86)
#16	first_aid (1), Health_Center (1), nursing_home (2), veterinary (15), childcare (22), dentist (22), doctors (27), clinic (38), social_facility (50), pharmacy (102), hospital (245)
#17	atm (1), police_station (1), office (71), police (72), post_office (88), fire_station (198), bank (213), fuel (289)
#18	shop (2), supermarket (6), retail (535)
#19	manufacture (7), industrial (573)
#20	bowling_alley (1), fitness_centre (1), gym (2), fitness_station (7), sports_centre (73), swimming_pool (932)
#21	company (1), studio (5), commercial (1,031)
#22	warehouse (129)
#23	town_hall (1), conference_centre (1), concert_hall (1), chapel 3), museum (4), arts_centre (17), public_building (21), courthouse (24), townhall (37), embassy (51), theatre (102), library (235)
#24	church (53), place_of_worship (1,783)
#25	ranger_station (2), water_park (9), dog_park (62), fountain (76), nature_reserve (105), garden (506), park (2,287)
#26	iona_college_dorm (3), music_school (1), convent (3), kindergarten (17), dormitory (53), college (96), university (290), school (2,297)
#27	bocce (2), horse_riding (2), shooting_stand (6), events_venue (7), grandstand (14), miniature_golf (19), bleachers (21), community_centre (31), recreation_ground (32), stadium (81), golf_course (88), playground (1,225), pitch (5,430)
#28	bandstand (1), food_and_drink (1), food_court (5), biergarten (6), ice_cream (8), ice_rink (13), pub (36), bar (75), fast_food (232), cafe (131), restaurant (670)
#29	bicycle_parking (8), parking_exit (1), parking_entrance (1), carport (5), parking_space (227), parking (6,867)
#30	mansion (2), apartment (2), residential (1,486), apartments (3,319)

TABLE 2
Statistics of the datasets.

POI dataset	Number of POIs	30,832
	Number of POIs with size	30,832
	Number of POIs with height	5,186
Graph dataset	Number of graph triples for training	5,000
	Number of graphs for testing	2,000

as Category #7 (i.e., venues for entertainment). We have removed highly frequent and non-distinctive POIs such as garages (101,561) and houses (17,035). We have also removed categories that have only one subcategory, and the subcategory has only one or two POIs (e.g., collapsed).

4.2 Experimental Setup

POI Datasets: We collect a POI dataset from New York City, USA. We obtain the POI categories, geo-coordinates, and sizes from OpenStreetMap. We merge the POIs into 30 categories based on 149 place type categories (cf. Table 1). We further obtain the POI heights (for buildings) from NYC Open Data[2]. We obtain the road network distances between POIs from the GraphHopper API[3]. We summarize the statistics of the collected POI dataset in Table 2.

We partition the New York City map (the highlighted area of the map in Figure 2) into a training area (80% of the highlighted area) and a testing area (the rest 20% of the highlighted area) without overlapping. We generate regions and graphs for model training based on this POI dataset following the procedure described in Section 3.1. For model testing, we generate 2,000 non-overlapping test graphs following the same steps as those for generating the graphs for model training. These graphs are also generated based on the POI dataset above.

Since there is no manually labeled ground-truth data, we generate three query graphs G_{q1}, G_{q2}, and G_{q3} for each testing graph G_t using the SGED-based algorithm with randomly generated noise ratio in the ranges of (0, 0.05], (0.05, 0.1], and (0.1, 0.15], respectively. We use G_t as the ground-truth similar graph of the three query graphs. Intuitively, since the testing graphs are non-overlapping, G_{q1}, G_{q2}, and G_{q3} should find G_t to be their most similar graph (or at least one of the most similar graphs) among the set of all testing graphs. Table 2 summarizes the statistics of the training and testing graph datasets.

Baselines. We compare with the following competitors.

SVSM [12]: This method uses a bags of POI strategy that represents a region with a distance-to-reference point matrix and computes region similarity by the cosine similarity of the matrices. For each region, the region center and four corner points are used as the reference points, and the average Euclidean distance of the POIs of each category to every reference point is stored in the distance-to-reference point matrix, i.e., a 30×5 matrix.

Triplet [1]: This method splits the whole map with a square grid, where each cell in the grid is represented by a vector corresponding to the POIs within the cell. The vector

2. https://opendata.cityofnewyork.us/
3. https://github.com/graphhopper/graphhopper

TABLE 3
Performance results of C-MPGCN, its variants, and competitors (the best results are in bold, the second best results are underlined).

	MRR			HR@1			HR@5			HR@10		
Noise ratio	0-0.05	0.05-0.1	0.1-0.15	0-0.05	0.05-0.1	0.1-0.15	0-0.05	0.05-0.1	0.1-0.15	0-0.05	0.05-0.1	0.1-0.15
SVSM [12]	90.68%	71.60%	60.67%	89.75%	68.60%	56.85%	91.60%	74.50%	64.40%	92.85%	77.60%	67.15%
Triplet [1]	68.17%	41.80%	39.26%	56.30%	33.95%	31.30%	83.70%	51.15%	47.15%	87.95%	54.90%	50.30%
CntrCNN [45]	79.85%	54.46%	38.18%	75.46%	45.23%	30.83%	86.10%	66.22%	47.08%	88.66%	70.61%	51.67%
GraphSim [30]	51.00%	21.58%	18.17%	35.25%	11.20%	9.90%	70.70%	30.40%	24.75%	82.90%	44.25%	34.10%
MVURE [15]	79.71%	60.78%	53.54%	70.56%	48.33%	35.56%	91.11%	76.66%	76.11%	97.78%	91.11%	84.44%
C-MPGCN	**99.28%**	**89.18%**	**77.58%**	**99.10%**	**87.00%**	**73.40%**	**99.55%**	**91.60%**	**81.90%**	**99.60%**	**93.40%**	**85.95%**
C-MPGCN-Hei	98.75%	86.60%	74.83%	98.35%	83.85%	70.75%	99.25%	89.45%	79.15%	99.45%	92.15%	82.60%
C-MPGCN-S	98.41%	85.07%	76.23%	98.10%	82.45%	72.60%	98.75%	88.35%	79.95%	99.10%	90.30%	83.80%
C-MPGCN-Hex	98.58%	84.26%	73.79%	98.25%	81.35%	69.70%	98.95%	87.10%	78.40%	99.15%	90.30%	81.80%
C-MPGCN-D	97.34%	84.98%	71.65%	97.00%	82.40%	67.45%	97.60%	87.50%	75.70%	98.05%	89.70%	80.25%
C-MPGCN-P	98.75%	84.53%	73.98%	98.50%	81.60%	70.05%	99.00%	88.10%	78.25%	99.20%	90.25%	81.50%
C-GCN	98.31%	86.59%	74.85%	97.85%	84.35%	70.90%	99.00%	88.65%	79.15%	99.10%	91.05%	82.30%

has a dimension of 30, and each value in the vector denotes the number of POIs in the corresponding category. Triplet first encodes the whole map into a large feature map. The region similarity of two cropped regions is then computed by the Euclidean distance between the corresponding sub feature maps within the learned whole feature map. Based on the Euclidean distance of different regions, a triplet contrastive learning-based CNN model was used to achieve the model training.

CntrCNN: This method splits a given region with a square grid, where each cell is represented by a vector corresponding to the POIs within the cell, like that in Triplet [1]. CntrCNN learns the embedding for each input region via a triplet contrastive learning-based CNN model [45]. After training the CntrCNN (the same training data as C-MPGCN), region similarity between two regions is computed by the Euclidean distance between their embedding vectors.

GraphSim [30]: This method addresses the problem of graph similarity computation by directly matching two sets of node embeddings without the need to use fixed-dimensional vectors to represent whole graphs. The nodes in each graph only have the category information, and the edge features are removed. We take the graph similarity score generated by our SGED-based algorithms as the ground-truth similarity score labels for training the GraphSim. After training, the Euclidean distance between graph embeddings is taken as the similarity when evaluating the model performance.

MVURE [15]: This method introduces a joint learning module that boosts the region embedding learning by sharing cross-view information and fuses multi-view embeddings by learning adaptive weights. We use the same training process as that in the paper. In the testing process, for each of the regions in the test dataset, we apply the idea of SGED to generate a similar region (e.g., adding a certain amount of noise: 0-0.05, 0.05-0.1, and 0.1-0.15 to the region features and region-wise relationships) as the ground truth region when doing the similar region search.

Evaluation metrics: We use two metrics to evaluate the effectiveness of C-MPGCN:

- MRR, which is a ranking-based metric where MRR = $1/rank$, and $rank$ denotes the rank of the ground-truth graph G_t in a model's returned list of a query graph G_q. We report the averaged MRR of all query graphs.
- HR@k, which denotes the averaged probability that a model returns the ground-truth graph G_t among its top-k answers given a query graph G_q.

Hyperparameters: MPGCN uses the following hyperparameters. In the embedding matrix learning module, the dimension of the learned embeddings after the first Multi-R GCN layer is 20, and that after the second Multi-R GCN layer is 10. The dimension of the output after the two GCN layers is then 30 because we concatenate the output of the two layers. In the assignment matrix learning module (pooling), we keep 10 nodes in its output. After pooling, MPGCN uses a fully connected layer to obtain the output embedding, which also has a dimension of 30. Batch normalization [46] is applied after each GCN layer, and L2 normalization is applied to the learned node embeddings. For the competitors SVSM and Triplet, we use the default settings from their original papers. Experiments are conducted on NVIDIA TESLA A100 GPUs with 80 GB memory.

The graphs generated for our C-MPGCN model and GraphSim are circular regions cropped from the training area with the radius γ of 1 km. For SVSM, Triplet, and CntrCNN, the corresponding inputs are square-shaped regions with a side length of 2 km sharing the same region centers as those used in C-MPGCN. The hexagon partition in C-MPGCN has a side length of 10 meters (the diameter of each cell's excircle is 20 meters), and the square partition in Triplet has a side length of 20 meters. The graphs generated for our C-MPGCN and competitors contain the same POI list but have different attribute values for each POI. The threshold θ for determining the neighboring road network distance in MPGCN is 400 meters.

4.3 Effectiveness

Table 3 shows the result on the collected NYC datasets. We see that C-MPGCN is substantially better than the competitors SVSM, Triplet, CntrCNN, GraphSim, and MVURE. In terms of MRR, C-MPGCN outperforms SVSM, Triplet, CntrCNN, GraphSim, and MVURE by 8.60%, 31.11%, 19.43%, 48.28%, and 19.57%, respectively, when the query graphs are generated via SGED-based algorithm with the noise ratio in the range of (0, 0.05]. With the increase of the

noise ratio to the range of (0.1, 0.15), the advantage of C-MPGCN over the three competitors increases to 16.91%, 38.32%, 39.40%, 59.41%, and 20.04%, respectively. In terms of HR@1, HR@5, and HR@10, similar performance gains are witnessed. Take HR@10 as an example. C-MPGCN outperforms SVSM, Triplet, CntrCNN, and MVURE by 6.75%, 11.65%, 10.94%, 63.85%, 16.70%, and 1.82%, respectively, when the query graphs are generated with the noise ratio in the range of (0, 0.05]. With the increase of the noise ratio to the range of (0.1, 0.15], the advantage of C-MPGCN generally increases to 18.80%, 35.65%, 34.28%, 51.85%, and 1.51%, respectively. These results confirm the effectiveness of C-MPGCN in capturing the region similarity. Regarding the inferior performance of GraphSim, we conjecture that this is because: (1) GraphSim is a graph matching algorithm where the edge features have been removed, while our problem is a graph similarity learning problem where both node and edge features play essential roles. (2) The datasets in the GraphSim paper contain graphs with less than 100 nodes, while that of our model contains hundreds and even over one thousand nodes. (3) GraphSim requires exact graph similarity scores for its model training, while the similarity scores computed by the SGED-based algorithm are fuzzy ones. Regarding the inferior performance of Triplet and CntrCNN to that of SVSM, we conjecture that the deep learning models Triplet and CntrCNN are impacted by the limited information in their training data and ineffective region representation. For these two models, their training data only contains POI category information, but not POI size and height information, which is needed to learn the region similarity under our problem setting. Also, they represent regions as grids instead of graphs. Their approach is less effective in region representation under our problem setting. Also, they represent regions as grids instead of graphs, which is more effective for region representation under our problem setting. SVSM, on the other hand, cannot fully explore the rich spatial information between POIs. Also, SVSM cannot capture the height and size information.

4.4 Ablation Study

To show the importance of different components of C-MPGCN, we further implement the following variants: **C-MPGCN-Hei**: C-MPGCN without capturing POI heights; **C-MPGCN-S**: C-MPGCN without capturing POI sizes; **C-MPGCN-Hex**: C-MPGCN without capturing POI hexagonal coordinates; **C-MPGCN-D**: C-MPGCN without discretizing the road network distances of neighboring POIs during MPGCN aggregation, and all neighbors are treated in a single road network distance category; **C-MPGCN-P**: C-MPGCN without pooling in MPGCN; **C-GCN**: C-MGPCN with MPGCN being replaced with a typical 2-layer GCN model [47].

Table 3 shows the performance results of these variants. All variants show inferior performance compared to the full C-MPGCN model. The inferior performance of C-MPGCN-Hei and C-MPGCN-S (which do not use POI heights and sizes) confirms that POI height and size play an important role in region similarity learning. This result also shows the importance of using SGED instead of GED, as SGED has an extra "soft replace" operation that helps capture POI height and size features.

The inferior performance of C-MPGCN-Hex, C-MPGCN-D (which do not consider the different road network distances between the POIs), and C-MPGCN-P (which does not consider POI importance) confirms that POI relationships such as isotropic properties, road network distances, and relative importance are also critical factors that should be considered in region similarity learning.

As for the C-GCN model, its inferior performance further confirms the effectiveness of the MPGCN component of our C-MPGCN model. In Table 3, all variants show superior performance compared to the baselines, which demonstrate the superiority of our graph representation and triplet generation methods.

4.5 Parameter Study

We study the impact of different pooling numbers, different θ values, and different γ values described in Equation 2.

The pooling number P_{num} denotes the number of nodes kept in the MPGCN model. We vary the value of P_{num} from 10 to 50 and summarize the results in Table 4. We set 50 as the upper bound as the average number of nodes within a graph in our collected dataset is between 50 and 60. When $P_{num} \geq 60$, the pooling process will not be able to capture the more important (i.e., representative) POIs. We observe that our model achieves the best performance when P_{num} is between 20 and 30, and we use 25 as the default setting.

Threshold θ in Equation 2 determines the neighboring POIs that C-MPGCN considers in aggregation. We set θ as 400 meters, which denotes that POIs within 400 meters are taken as neighbors, and the neighbors are separated into two categories, 0-200 meters, and 200-400 meters. We vary θ among [200, 400, 800, 1,600, 3,200]. Since the cropped regions have a radius of 1 km, all POIs within a region will be taken as neighbors when θ is 3,200 meters. We summarize the results in Table 5, and observe that our model achieves the best performance when θ is 400. A larger θ negatively impacts the model performance, because there are too many neighbors for each node. During the message passing process, different nodes aggregate information from similar sets of nodes and obtain similar node representations, which will lead to the over smoothing problem [48]. A small θ also brings down the model performance, because there are too few or even no neighbors to be considered, such that little information can be gained from message passing.

Parameter γ denotes the radius of the regions that we crop for training and testing the C-MPGCN model. A small γ leads to few POIs within each region. In this case, the negative regions generated by the SGED-based algorithm may be much different from the original graph and hence contain less information to learn. On the other hand, a large γ leads to many POIs from different categories residing in each region. In this case, the regions may be more difficult to differentiate since many regions contain POIs from all kinds of categories. We vary the value of γ from 0.5 km to 2 km for model training and testing, and we summarize the results after 200 training epochs with a batch size of 45 in Table 6. We observe that the performance of our model improves when γ increases from 0.5 km to 1 km. When γ continues to increase and exceeds 1.5 km, the model performance decreases. These results are consistent with our analysis above. We thus have used $\gamma = 1$ km as our default setting.

TABLE 4
Performance results of C-MPGCN with different values of P_{num} (the best results are in bold, and the second-best results are underlined).

	MRR			HR@1			HR@5			HR@10		
Noise ratio	0-0.05	0.05-0.1	0.1-0.15	0-0.05	0.05-0.1	0.1-0.15	0-0.05	0.05-0.1	0.1-0.15	0-0.05	0.05-0.1	0.1-0.15
P_{num} = 10	98.78%	84.72%	75.11%	98.40%	81.10%	71.50%	99.25%	87.80%	78.75%	99.35%	90.60%	81.85%
P_{num} = 20	99.06%	**88.13%**	**77.98%**	98.85%	**85.95%**	**74.20%**	**99.50%**	90.40%	**81.95%**	**99.70%**	91.90%	**84.95%**
P_{num} = 30	**99.28%**	87.56%	77.57%	**99.10%**	84.20%	73.95%	99.45%	**91.25%**	81.50%	99.55%	**93.80%**	84.10%
P_{num} = 40	98.44%	83.61%	74.22%	98.05%	80.00%	70.15%	98.95%	88.15%	78.90%	99.10%	90.65%	82.15%
P_{num} = 50	98.18%	84.65%	72.82%	97.80%	81.90%	68.60%	98.70%	87.55%	76.65%	98.85%	90.40%	80.90%

TABLE 5
Performance results of C-MPGCN with different θ values (the best results are in bold).

	MRR			HR@1			HR@5			HR@10		
Noise ratio	0-0.05	0.05-0.1	0.1-0.15	0-0.05	0.05-0.1	0.1-0.15	0-0.05	0.05-0.1	0.1-0.15	0-0.05	0.05-0.1	0.1-0.15
θ = 200	98.65%	85.23%	73.54%	98.20%	82.75%	69.90%	99.10%	87.80%	77.25%	99.35%	90.25%	80.30%
θ = 400	**99.28%**	**89.18%**	**77.58%**	**99.10%**	**87.00%**	**73.40%**	**99.55%**	**91.60%**	**81.90%**	**99.60%**	**93.40%**	**85.95%**
θ = 800	97.22%	81.22%	74.20%	96.70%	77.90%	70.40%	97.70%	84.55%	78.05%	98.25%	87.35%	81.90%
θ = 1600	98.40%	77.32%	69.49%	97.70%	72.45%	64.10%	98.90%	83.00%	75.20%	99.35%	86.65%	80.05%
θ = 3200	97.34%	84.98%	71.65%	97.00%	82.40%	67.45%	97.60%	87.50%	75.70%	98.05%	89.70%	80.25%

TABLE 6
Performance results of C-MPGCN with different γ values (the best results are in bold, and the second-best results are underlined).

	MRR			HR@1			HR@5			HR@10		
Noise ratio	0-0.05	0.05-0.1	0.1-0.15	0-0.05	0.05-0.1	0.1-0.15	0-0.05	0.05-0.1	0.1-0.15	0-0.05	0.05-0.1	0.1-0.15
γ = 0.5 km	95.53%	77.73%	68.42%	95.15%	74.80%	64.30%	95.95%	80.85%	72.50%	96.50%	84.30%	75.85%
γ = 1 km	96.80%	**82.52%**	71.36%	96.10%	**79.50%**	67.30%	97.60%	**86.15%**	**76.15%**	97.95%	88.35%	**79.80%**
γ = 1.5 km	**97.63%**	81.87%	**71.93%**	**97.15%**	78.60%	**68.50%**	**98.25%**	85.10%	75.20%	**98.50%**	**88.80%**	78.70%
γ = 2 km	92.62%	72.29%	58.38%	91.75%	69.00%	53.80%	93.55%	75.65%	62.95%	94.45%	78.65%	66.20%

4.6 Efficiency and Complexity

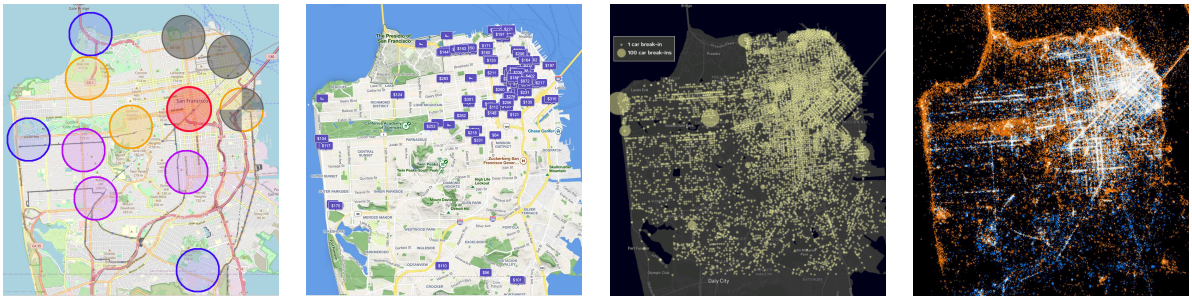
The efficiency of C-MPGCN. To evaluate the running time efficiency of C-MPGCN and the baseline models, we run Triplet, CntrCNN, C-MPGCN, GraphSim, and MVURE with the same batch size 45 and summarize the averaged training times per batch in Table 7. SVSM does not have a training time because it compares the Euclidean distance of the feature vectors of two regions. Among the other models, MVURE has the lowest training time complexity due to its simple model structure. The training time of C-MPGCN is on par with those of Triplet, CntrCNN, and GraphSim i.e., all four models can train on a batch with about 6 seconds.

TABLE 7
Running Time Efficiency of C-MPGCN and the baselines.

	Training	Testing		
		0-0.05	0.05-0.1	0.1-0.15
SVSM [12]	N/A	0.47 s	0.41 s	0.41 s
Triplet [1]	6.10 s	0.14 s	0.13 s	0.13 s
CntrCNN [45]	6.85 s	0.07 s	0.07 s	0.07 s
GraphSim [30]	6.04 s	13.41 s	12.32 s	12.06 s
MVURE [15]	0.03 s	0.014 s	0.015 s	0.016 s
C-MPGCN	6.52 s	0.20 s	0.14 s	0.14 s

To further evaluate the inference time efficiency, we run C-MPGCN and the baseline models for 2,000 region similarity queries using different noise ratios (randomly chosen within the ranges of (0, 0.05], (0.05, 0.1], and (0.1, 0.15], respectively). Table 7 reports the averaged query times for each query. Among the six models, GraphSim is the most time-consuming, while MVURE is the fastest again due to its simple model structure. The inference time of C-MPGCN is on par with that of Triplet and is around twice that of CntrCNN. Considering the performance gain of C-MPGCN, we argue that its extra time costs are worthwhile.

The complexity of SGED. Computing the exact GED is NP-complete [21]. For our graph generation algorithm using the SGED-based algorithm, suppose that there are $nc(G_r)$ nodes in a target graph G_r for which a positive (or negative) sample (i.e., a graph variant) is to be generated, and there are n nodes in the entire training dataset. The time complexity of the four graph edit operations in SGED are: (1) $O(1)$ for the "add" operation, as we randomly select a POI from the entire POI set, (2) $O(1)$ for the "remove" operation, as we randomly select a POI from the target graph, (3) $O(1)$ for the "soft replace (Type I)" operation, as we randomly select a POI from the entire POI set to replace a randomly selected POI from the target graph, and (4) $O(1)$ for the "soft replace (Type II)" operation with the same reason for the "remove" operation, respectively. Our graph generation algorithm based on SGED uses a sequence of SGED edit operations to generate a graph similar or dissimilar to the target graph G_r according to a given similarity threshold $\epsilon \in (0, 1)$. The average edit distance of each operation is $O(1/nc(G_r))$ because there are $nc(G_r)$ nodes in the target graph. Therefore, the complexity of the averaged number of



(a) (b) (c) (d)

Fig. 6. (a) A case study with an area in San Francisco (the red region is the query region; the gray, orange, purple, and blue regions are the top-3 similar regions returned by C-MPGCN, CntrCNN, Triplet, and SVSM, respectively); (b) Hotel distribution; (c) Car break-ins distribution; (d) Twitter usage distributions.

edit operations to reach ϵ is bounded by the $O(nc(G_r))$. The overall time complexity of our graph generation algorithm is $O(nc(G_r))$.

4.7 Case study

To further show the effectiveness of our C-MPGCN model, we apply the trained C-MPGCN model based on the NYC dataset to a case study using a map area in San Francisco. We select a POI in San Francisco and crop a query region r with a radius of 1 km around the POI, which is denoted by the red circled region in Figure 6. The trained models (including our C-MPGCN, CntrCNN, Triplet, and SVSM) then search the regions within San Francisco to find regions that are similar to r without overlapping with it.

Figure 6a shows the top-3 similar regions identified by C-MPGCN (the gray ones), CntrCNN (the orange ones), Triplet (the purple ones), and SVSM (the blue ones), respectively. "⊕" denotes that the resulting region is shared by both C-MPGCN and CntrCNN. Figure 6b to Figure 6d show the distribution of hotel, car break-ins[4], and Twitter usage[5] in San Francisco, which are used to help benchmark the similar regions identified by different methods (C-MPGCN, CntrCNN, Triplet, and SVSM). We can see that the similar regions identified by C-MPGCN are those that share a higher similarity with r in Figures 6b to 6d. Compared with the baseline methods, the results showcase the effectiveness of C-MPGCN.

5 Conclusion

We proposed a triplet contrastive learning-based graph convolutional neural network model named C-MPGCN for region similarity learning, where a region is represented as a graph. To generate similar and dissimilar graph triples for model training, we propose a soft graph edit distance-based algorithm that measures the similarity between two graphs (i.e., regions) according to a modified edit distance from one graph to the other. From the generated data, C-MPGCN learns to capture region features based on POIs, including POIs' spatial features and relationships such as their road network distances. C-MPGCN also learns graph structural information such as a node hierarchy via a pooling-based multi-relational GCN module. Experiments show that C-MPGCN outperforms state-of-the-art methods for region similarity learning consistently.

4. https://projects.sfchronicle.com/2018/sf-car-breakins/
5. https://a.fastcompany.net/upload/San-FranciscoBig.jpg

6 Acknowledgements

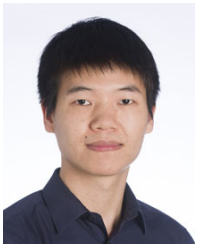
We thank Khoa D. Doan for the valuable discussion when preparing this manuscript. This work is partially supportedby the National Natural Science Foundation of China (No. 32070025) and the Australian Research Council (ARC) Discovery Project (DP230101534).

References

[1] Y. Liu, K. Zhao, and G. Cong, "Efficient similar region search with deep metric learning," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018, pp. 1850–1859.
[2] Y. Liu, T.-A. N. Pham, G. Cong, and Q. Yuan, "An experimental evaluation of point-of-interest recommendation in location-based social networks," *Proceedings of the VLDB Endowment*, vol. 10, no. 10, pp. 1010–1021, 2017.
[3] T.-A. N. Pham, X. Li, and G. Cong, "A general model for out-of-town region recommendation," in *The Web Conference*, 2017, pp. 401–410.
[4] G. McKenzie and D. Romm, "Measuring urban regional similarity through mobility signatures," *Computers, Environment and Urban Systems*, vol. 89, p. 101684, 2021.
[5] S. Ashkezari-Toussi, M. Kamel, and H. Sadoghi-Yazdi, "Emotional maps based on social networks data to analyze cities emotional structure and measure their emotional similarity," *Cities*, vol. 86, pp. 113–124, 2019.
[6] H. Wang, D. Kifer, C. Graif, and Z. Li, "Crime rate inference with big data," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 635–644.
[7] H. Yao, F. Wu, J. Ke, X. Tang, Y. Jia, S. Lu, P. Gong, and J. Ye, "Deep multi-view spatial-temporal network for taxi demand prediction," in *AAAI Conference on Artificial Intelligence*, 2018, pp. 2588–2595.
[8] S. Liu, Q. Liu, and Z. Bao, "Dars: Diversity and distribution-aware region search," in *International Conference on Database Systems for Advanced Applications*, 2020, pp. 204–220.
[9] X. Jin, B. Oh, S. Lee, D. Lee, K.-H. Lee, and L. Chen, "Learning region similarity over spatial knowledge graphs with hierarchical types and semantic relations," in *ACM International Conference on Information and Knowledge Management*, 2019, pp. 669–678.
[10] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Annual Conference on Neural Information Processing Systems*, 2018, pp. 4805–4815.
[11] G. Le Falher, A. Gionis, and M. Mathioudakis, "Where is the soho of rome? measures and algorithms for finding similar neighborhoods in cities." *International AAAI Conference on Web and Social Media*, vol. 2, pp. 228–237, 2015.

[12] C. Sheng, Y. Zheng, W. Hsu, M. L. Lee, and X. Xie, "Answering top-k similar region queries," in *International Conference on Database Systems for Advanced Applications*, 2010, pp. 186–201.
[13] B. Kulis *et al.*, "Metric learning: A survey," *Foundations and Trends in Machine Learning*, vol. 5, no. 4, pp. 287–364, 2012.
[14] B. Wang, Y. Lin, S. Guo, and H. Wan, "Gsnet: Learning spatial-temporal correlations from geographical and semantic aspects for traffic accident risk forecasting," in *AAAI Conference on Artificial Intelligence*, 2021, pp. 4402–4409.
[15] M. Zhang, T. Li, Y. Li, and P. Hui, "Multi-view joint graph representation learning for urban region embedding," in *International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 4431–4437.
[16] B. Hui, D. Yan, W.-S. Ku, and W. Wang, "Predicting economic growth by region embedding: A multigraph convolutional network approach," in *ACM International Conference on Information & Knowledge Management*, 2020, pp. 555–564.
[17] X. Gao, B. Xiao, D. Tao, and X. Li, "A survey of graph edit distance," *Pattern Analysis and Applications*, vol. 13, no. 1, pp. 113–129, 2010.
[18] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern Recognition Letters*, vol. 19, no. 3-4, pp. 255–259, 1998.
[19] K. D. Doan, S. Manchanda, S. Mahapatra, and C. K. Reddy, "Interpretable graph similarity computation via differentiable optimal alignment of node embeddings," in *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021, pp. 665–674.
[20] B. H., "Error correcting graph matching: on the influence of the underlying cost function," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 9, pp. 917–922, 1999.
[21] Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou, "Comparing stars: On approximating graph edit distance," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 25–36, 2009.
[22] D. B. Blumenthal and J. Gamper, "On the exact computation of the graph edit distance," *Pattern Recognition Letters*, vol. 134, pp. 46–57, 2020.
[23] X. Zhao, C. Xiao, X. Lin, Q. Liu, and W. Zhang, "A partition-based approach to structure similarity search," *Proceedings of the VLDB Endowment*, vol. 7, no. 3, pp. 169–180, 2013.
[24] Y. Liang and P. Zhao, "Similarity search in graph databases: A multi-layered indexing approach," in *IEEE International Conference on Data Engineering*, 2017, pp. 783–794.
[25] S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gaüzère, and M. Vento, "Graph edit distance as a quadratic assignment problem," *Pattern Recognition Letters*, vol. 87, pp. 38–46, 2017.
[26] S. Fankhauser, K. Riesen, and H. Bunke, "Speeding up graph edit distance computation through fast bipartite matching," in *International Workshop on Graph-Based Representations in Pattern Recognition*, 2011, pp. 102–111.
[27] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, "Simgnn: A neural network approach to fast graph similarity computation," in *ACM International Conference on Web Search and Data Mining*, 2019, pp. 384–392.
[28] L. Chang, X. Feng, X. Lin, L. Qin, W. Zhang, and D. Ouyang, "Speeding up ged verification for graph similarity search," in *International Conference on Data Engineering*, 2020, pp. 793–804.
[29] R. E. Burkard, E. Çela, P. M. Pardalos, and L. S. Pitsoulis, *The Quadratic Assignment Problem*, 1997.
[30] Y. Bai, H. Ding, K. Gu, Y. Sun, and W. Wang, "Learning-based efficient graph similarity computation via multi-scale convolutional set matching," in *AAAI Conference on Artificial Intelligence*, 2020, pp. 3219–3226.
[31] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *International Conference on Machine Learning*, 2019, pp. 3835–3845.
[32] R. Wang, T. Zhang, T. Yu, J. Yan, and X. Yang, "Combinatorial learning of graph edit distance via dynamic embedding," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5241–5250.
[33] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.
[34] N. Wu, X. W. Zhao, J. Wang, and D. Pan, "Learning effective road network representation with hierarchical graph neural networks," in *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 6–14.
[35] J. Sun, J. Zhang, Q. Li, X. Yi, Y. Liang, and Y. Zheng, "Predicting citywide crowd flows in irregular regions using multi-view graph convolutional networks," *IEEE Transactions on Knowledge and Data Engineering*, 2020.
[36] L. Liu, M. Liu, G. Li, Z. Wu, and L. Lin, "Road network guided fine-grained urban traffic flow inference," *arXiv preprint arXiv:2109.14251*, 2021.
[37] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.
[38] W. Schneider, "Bbbike extracts openstreetmap," Dec. 2018. [Online]. Available: https://extract.bbbike.org/
[39] J. Ke, H. Yang, H. Zheng, X. Chen, Y. Jia, P. Gong, and J. Ye, "Hexagon-based convolutional neural network for supply-demand forecasting of ride-sourcing services," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 11, pp. 4160–4173, 2019.
[40] I. Her, "Geometric transformations on the hexagonal grid," *IEEE Transactions on Image Processing*, vol. 4, no. 9, pp. 1213–1222, 1995.
[41] L.-y. Sun, C.-l. Miao, and L. Yang, "Ecological-economic efficiency evaluation of green technology innovation in strategic emerging industries based on entropy weighted topsis method," *Ecological Indicators*, vol. 73, pp. 554–558, 2017.
[42] A. Lesne, "Shannon entropy: A rigorous notion at the crossroads between probability, information theory, dynamical systems and statistical physics," *Mathematical Structures in Computer Science*, vol. 24, no. 3, 2014.
[43] X. Miao, W. Zhang, Y. Shao, B. Cui, L. Chen, C. Zhang, and J. Jiang, "Lasagne: A multi-layer graph convolutional network framework via node-aware deep architecture," *IEEE Transactions on Knowledge and Data Engineering*, 2021.
[44] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Annual Conference on Neural Information Processing Systems*, 2017, pp. 1024–1034.
[45] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.
[46] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.
[47] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2016.
[48] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *AAAI Conference on Artificial Intelligence*, vol. 34, no. 4, 2020, pp. 3438–3445.

Yunxiang Zhao is an assistant professor at Beijing Institute of Biotechnology. He received his Ph.D. degree from The University of Melbourne. His research interests include spatial data analysis, bioinformatics, and deep learning.



Jianzhong Qi is a Senior Lecturer in the School of Computing and Information Systems at The University of Melbourne. He received his Ph.D. degree from The University of Melbourne in 2014. His research interests include machine learning and data management and analytics, with a focus on spatial, temporal, and textual data.

Bayu D. Trisedya is a Lecturer in the Faculty of Computer Science Universitas Indonesia, who is currently a Postdoctoral Research Fellow at RMIT University. He received bachelor's and master's degrees from Universitas Indonesia in 2009 and 2011, respectively and the Ph.D. degree in computer science from The University of Melbourne in 2021. His research interests include information extraction and NLP.

Yixin Su is a Ph.D. candidate in the School of Computing and Information Systems at University of Melbourne. His research interests include graph neural network based recommender systems and deep learning.

Rui Zhang is an internationally leading researcher in the area of big data, data mining, and machine learning. His research interests include big data, data mining, and machine learning. He has won several awards including Future Fellowship by the Australian Research Council in 2012, and Google Faculty Research Award in 2017.

Hongguang Ren is an associate professor at Beijing Institute of Biotechnology. His research interests include bioinformatics, genomic epidemiology, and deep learning.