Path2Vec: Representation Learning for Node Sequences Based on Contrastive Learning

Takayasu Fushimi[1][0000−0003−3448−8182] and Yuki Kawasaki[1]

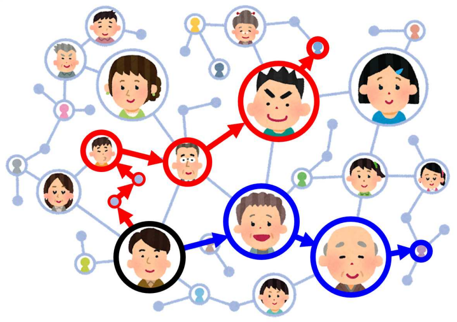
Tokyo University of Technology, Tokyo, Japan
{fushimity,c0b200453f}@edu.teu.ac.jp

Abstract. We often deal with sequences of nodes on graphs, such as routes from departure points to destinations in road networks, and information diffusion routes from information sources in SNS. While there has been much research on nodes and edges in graph machine learning, and many methods for embedding them in vector spaces have been proposed, there is little research on node sequences (path), and a method for embedding node sequences in vector space has not been established. In this study, for node sequences, we propose a novel method for embedding paths in a vector space by considering the nature and appearance order of nodes. Our method learns the vectors so that paths consisting of nodes with similar characteristics or consisting of the same nodes become similar vectors, whereas consisting of nodes with different properties or distant nodes become dissimilar vectors. The proposed method uses the framework of Contrastive Learning, which is a type of self-supervised learning. Specifically, a similar node sequence is generated by fixing the starting point, intermediate points, and ending point, and data-augmenting the path based on a random walk. Then, we nonlinearly project the encoded vectors considering the node vectors and the order of appearance, and learn various parameters so that the similarity of the embedding vectors of node sequences generated from the same starting point and ending point is high. Through evaluation experiments using actual road network data, we confirm that the desired vector can be obtained.

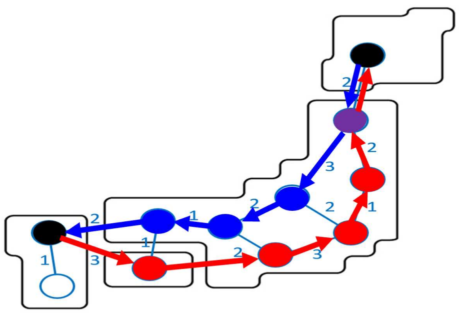
Keywords: Embedding · Node Sequences · Contrastive Learning · Data Augmentation · Random Walk

1 Introduction

In recent years, artificial intelligence services have become available and used to perform tasks in various fields, such as text generation, automatic composition/singing, and image generation. Machine learning, the core technology of artificial intelligence, can learn the rules of various data such as texts, images, audio, music, and graphs. The relationships between various things observed in the real world can be expressed in graphs. For example, following relationships in social networks such as X (formerly Twitter), Facebook, and Instagram can be expressed as a graph with users as nodes and following/follower relationships



(a) information diffusion path on a social graph



(b) traffic route on a road graph

Fig. 1. Example of node sequences

as edges. Similarly, hyperlink relationships between web pages on the Internet can be expressed as a graph with web pages as nodes and hyperlinks as edges. Many studies have been conducted on machine learning for graphs, and the main tasks include node classification to predict the category and label of each node, link prediction to predict unknown edges and links on a graph, and examples include graph generation, which generates new graphs based on specified conditions. Since many machine learning methods require vectors as input, it is necessary to embed them in a vector space, and many methods related to node embedding and edge embedding have been proposed [1]. On the other hand, there are also studies that target paths, which are sequences of nodes, rather than single nodes or edges. For example, in research on information diffusion sequences on SNS, it is important to know what kind of nodes are interested in certain information, how it cascades, and how the information spreads. As another example, in research on route recommendation systems on road networks, it is important to recommend not only the shortest route from a departure point to a destination, but also a route that avoids traffic congestion and emphasizes scenery. In research targeting such node sequences, it is necessary to calculate the similarity or distance between node sequences by considering the properties of the included nodes and the order of the nodes. In the example of diffusion paths on SNS in Fig. 1(a), for different information from the same user, the red line shows that only men react and information flows, and the blue line shows that only elderly people react and information flows. In this case, the nature of the node is the user's demographic attributes such as age and gender. Although they are similar in that the variation in node properties within each sequence is small, the properties of the nodes between sequences are very different. In the example of traffic routes on the road network in Fig. 1(b), the blue route goes south along the coast from northeast to southwest, and the red route goes north along the coast. If the properties of the nodes are latitude and longitude, these sequences become dissimilar sequences. On the other hand, if the properties of the nodes are landscape, the sequences are similar, although the cities pass through is different. In these examples, the characteristics of the nodes are demographic attributes, latitude and longitude, and scenery, but depending on the purpose, it is also possible to represent them with vectors that take into account adjacency relationships using node2vec[2], etc.

Therefore, finding embedding vectors that take these information into consideration is an important research topic. However, while there are many studies on obtaining embedding vectors for nodes and edges, there are not many studies on embedding node sequences. In this study, we propose a path embedding method, which we call path2vec, that considers the properties and appearance order of nodes on the node sequence. Our method embeds sequences in which nodes appear in a similar order into similar vectors, while sequences in which the nodes have different properties, and sequences in which the nodes that appear have the same properties but appear in different orders, are embedded in dissimilar vectors. However, each node does not have to be the same one because it is expressed by a vector based on external information such as demographic attributes or latitude and longitude, or a vector that reflects graph properties such as node2vec.

Seq2vec [3] is a typical method for vector representation of sequences, but it cannot be applied to the node sequence targeted in this research because it requires a supervised label for each element of the sequence or the entire sequence. Our method employs the framework of contrastive learning, which is a type of self-supervised learning [4], which is attracting attention in the fields of image processing and natural language processing because it has achieved high accuracy comparable to supervised learning. Specifically, a similar node sequence is generated by fixing the starting point, intermediate points, and ending point, and expanding the node sequence based on a random walk. Then, we nonlinearly project the encoded vectors considering the node vectors and the order of appearance, Various parameters are learned so that the similarity of the embedding vectors of node sequences generated from the same starting point and ending point is high.

The technical novelties in the proposed method are 1) the ability to embed node sequences (paths) without teacher labels, 2) the ability to control the degree of similarity with the original sample to some extent by data augmentation based on random walk, and 3) the ability to express the diversity of similarity of node sequences by using modified NTXent loss.

The remain of this paper is organized as follows: Section 2 introduces related research. Section 3 describes the proposed methods, and Sections 4 and 5 present evaluation experiments. Finally, Section 6 summarizes this study and mentions future work.

2 Related Work

2.1 Graph Embedding

Embedding nodes and graphs has been widely studied for many years. A classical approach is a method called spectral clustering that uses the eigenvectors of the

Laplacian matrix [5]. This method is known to be effective for clustering, and has been proven to be effective for graphs in which communities are sufficiently separated. Also, since it is a linear method, it has the advantage of being relatively fast to execute. Examples of nonlinear methods include the spring model [6, 7] and the cross-entropy method [8]. The former calculates node coordinates so that the graph distance and the distance between embedded vectors match as much as possible. The latter focuses on embedding pairs of adjacent nodes closer together in the graph.

In recent years, DeepWalk [9], Node2Vec [2], LINE [10], GraRep [11], SDNE [12] are representative and often cited as comparison methods. The purpose of these methods is not to visualize graphs, but to express the structural characteristics of nodes as vectors. For example, DeepWalk and Node2Vec generate a sequence of nodes by performing a random walk starting at each node. The method applies Skip-Gram or CBoW to node sequences in order to obtain the representation vectors of the nodes. On the other hand, GraRep decomposes the adjacency matrix and obtains a representation vector in order to consider not only directly adjacent nodes but also nodes two or three hops away. In addition to this, a method called edge2vec has been proposed as a method for edge embedding[13]. In the method proposed in this study, the characteristics of nodes that appear in a node sequence are expressed as vectors in the encoder layer. The above-mentioned method is effective when expressing features based on adjacency relationships in a graph. In the experiment, we use GraRep to calculate the representation vectors of the nodes included in the node string.

Although order relationships are ignored, hypergraph embedding methods that treat a collection of multiple nodes as hyperedges are also being actively researched. As an embedding method for hypergraphs, a method based on the Laplacian matrix of the clique-expanded graph has been proposed [14]. Although it adjusts the degree of the node after clique expansion, it is essentially equivalent to the above-mentioned spectral clustering and laplacian eigenmap, As shown in the evaluation experiments in this paper, there are cases where the results are of poor quality due to the limitations of the linear method.

In a method called NetVec proposed in [15], the hypergraph is converted into a bipartite graph, the vector of the hyperedge is defined by composing the vectors of the adjacent hypernode, and the vector of the hypernode is defined by composing the vectors of the adjacent hyperedges. Although NetVec is common with our method in that the vectors of hypernodes and hyperedges are fixed and updated alternately and these update processes are iteratively repeated, NetVec differs in that vectors are not normalized by their norm and that it considers the vector one iteration before by a certain rate.

As a method for learning the embedding representation of hypergraphs derived from Location Based Social Networks, LBSN2Vec has been proposed [16]. In LBSN2Vec, as with Node2Vec, the relationship between hypernodes is regarded as sequence data based on a random walk on the graph, and an embedding representation is acquired. When learning the embedding vector, the regression line that maximizes the cosine similarity with each node vector is obtained from the composite vector of the representation vectors of the nodes included in a certain hyperedge. Then, the embedding vector is learned so that the cosine similarity is the maximum for these nodes and the cosine similarity is the minimum for the negatively sampled nodes. It is common with our method in that the embedding vector is obtained so that the cosine similarity between the vectors is maximized, but it differs from this study in that it is sequenced by a random walk and negative sampling is used. Our method attempts to embed both nodes and edges into the same spherical surface as these methods [15, 16] do.

HGNN (Hypergraph Neural Networks) and its generalization, HGNN+, are neural network architectures designed to address node classification problems on hypergraphs [17, 18]. Specifically, they utilize a specialized tensor structure to represent hypergraphs. In hypergraphs, multiple nodes can simultaneously share an edge, requiring a specific tensor structure to capture the hypergraph representation effectively. This tensor is shared across hyperedges of different sizes and captures important information. Using this tensor, convolutional operations are performed to update node features, and node classification is carried out using these features. The main contribution of HGNN+ is providing a neural network architecture for problems on hypergraphs. This architecture can be applied to different types of graphs and demonstrates high performance. Moreover, it can handle different types of hyperedges, allowing for flexible application depending on the problem at hand.

2.2 Contrastive Learning

Contrastive learning is a framework classified as self-supervised learning. In recent years, this technology has attracted attention not only in image processing but also in the field of natural language processing, as it has achieved the same level of accuracy as supervised learning. As a contrastive learning method for image classification tasks, there is a method called SimCLR by Chen et al. [19]. SimCLR performs spatial transformations such as cropping + resizing (+ flipping), rotation, and cutout on input images. Generate similar images by performing data augmentation, which applies visual transformations such as color distortion, Gaussian blur, and Sobel filters. ResNet50, which is a typical example of an image feature extractor, encodes input images and data-augmented images into feature vectors. Then, project it to a nonlinear vector using the Projection Head. The similarity between the embedding vectors of the original image and the data-augmented image is increased. Learn the parameters of Encoder and Projection Head.

Even though SimCLR has a simple architecture, it is possible to build highly accurate models using large batches. However, there are many cases where large batch sizes become a bottleneck. To solve this problem, there is a method called PIRL that stores feature vectors in a dictionary called a memory bank [20]. In PIRL, negative examples are randomly sampled from a memory bank, which eliminates the need for large batch sizes. However, it is necessary to update the feature vectors in the memory bank as appropriate, and updating requires a large amount of cost. In order to alleviate this problem, a method called MoCo, which introduces a mechanism called a momentum encoder, has been proposed [21]. In MoCo, only positive examples are passed through the momentum encoder, and negative examples use embedded representations of past positive examples.

As another architecture, a method called SwAV has been proposed that uses clustering to alleviate the problem of learning negative examples even if they contain the same label as the original sample[22].

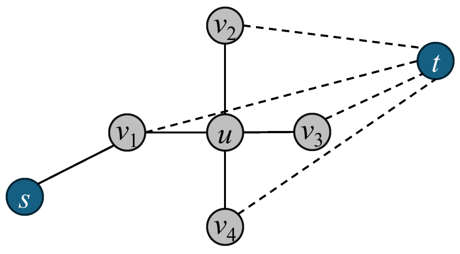
A method called HyperGCL has also been proposed to embed hypergraphs in a contrastive learning framework [23]. In HyperGCL, hyper-edges are deleted or some hyper-nodes are deleted from hyper-edges as augmentation to the hypergraph. It then acquires an embedded representation of the hypergraph in the framework of a Variational Auto-Encoder. As with the hypergraph embedding methods described above, these methods are not applicable to node sequences in which the same node may appear multiple times and may contain more nodes than hyperedges. In addition, they cannot be used for the purpose of this study in that they ignore the order information of node sequences whose order is meaningful.

In this study, we apply the SimCLR framework, which is contrastive learning, to convert nodes with similar features in a graph or node sequences consisting of the same nodes into similar vectors, and nodes with different features or different nodes into similar vectors. Representations are learned so that the columns become dissimilar vectors.

3 Proposed Method

In this study, for a graph $G = (\mathcal{V}, \mathcal{E})$ consisting of a node set $\mathcal{V}$ and an edge set $\mathcal{E}$, we propose a method to obtain the embedding vector of the node sequence $r = [u \in \mathcal{V}]$. In our method, a positive example (pos) of an input node sequence is generated by random-walk-based data augmentation and whose embedding is learned to locate in a neighbor in vector space, while the other samples are treated as negative ones (neg) and whose embeddings are learned to be located far away. Furthermore, we introduce a similar node sequence (sim) that shares a part of the node sequence and a node sequence that is the exact opposite of the input (opp). Then, it learns so that the similarity with the original is $+1 >$ pos $>$ sim $>$ neg $>$ opp > -1. An overview of the proposed method is shown below:

1. Data Augmentation: For $r \in \mathcal{R}$, by fixing the starting point $s \leftarrow r[0]$ and the ending point $t \leftarrow r[-1]$, generating the node sequence $r^{(1)}, r^{(2)}$ by applying a biased random walk;
2. Encoder: For r, calculating the weighted average vector of node embeddings $\mathbf{x}_u \leftarrow \text{Embedding}(u; \theta_e)$, weighted by the order of the node in the sequence $\mathbf{y}_r \leftarrow \sum_{u \in r} \text{Weight}(u; \theta_w)\mathbf{x}_u$;
3. Projection Head: For r, by performing the nonlinear transformation, calculating the embedding vector $\mathbf{z}_r \leftarrow \text{Projector}(\mathbf{y}_r; \theta_p)$;



$$d(v_1 \to t) > d(v_4 \to t) > d(v_2 \to t) > d(v_3 \to t)$$

$$p(u \to v_1) < p(u \to v_4) < p(u \to v_2) < p(u \to v_3)$$

Fig. 2. Biased random walk

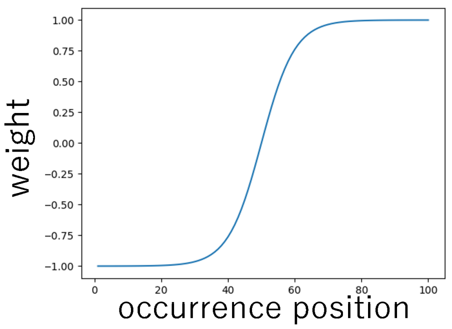


Fig. 3. Weight for the occurrence position.

4. NTXent loss: Using the embedding vector $\mathbf{z}_r$ from the proposed model, calculating the similarity for each positive, similar, negative, and opposite pair, and the modified Normalized Temperature-Scaled Cross-Entropy loss value.

The backpropagation is performed on the modified NTXent loss value to learn various parameters of the proposed model. Then, by inputting the node sequence r' to the trained model, the embedding vector $\mathbf{z}'$ is obtained. The details of each step will be explained in the following subsections.

3.1 Data Augmentation

For a given node sequence $r \in \mathcal{R}$ as input, by fixing the starting point $s \leftarrow r[0]$ and the ending point $t \leftarrow r[-1]$, we generate a path by a random walk with a bias from the starting point to the ending point:

$$r' \leftarrow \mathrm{RandomWalk}(r[0], r[-1]; \mu).$$

In a random walk, the transition probability from node u to its neighbor $v \in \Gamma(u)$ is defined as follows (also see Fig 2):

$$p(u \to v) = \frac{\exp(-\mu d(v \to t))}{\sum_{v' \in \Gamma(u)} \exp(-\mu d(v' \to t))},$$

where $d(v \to t)$ is the distance from the adjacent node v to the endpoint node t and μ is a parameter that controls the bias: the larger the value of μ, the stronger the bias toward the endpoint t, and the smaller the value of μ, the smaller the probability difference between adjacent nodes $v \in \Gamma(u)$ and the more random the walk becomes.

In this study, we generate a positive sample pos_r by concatenating two generated paths each of which is generated by fixing starting $r[0]$ and middle points $r[\mathrm{mid}]$, and middle $r[\mathrm{mid}]$ and ending points $r[-1]$:

$$\begin{aligned}\mathrm{pos}_r \leftarrow [&\mathrm{RandomWalk}(r[0], r[\mathrm{mid}]; \mu_1);\\ &\mathrm{RandomWalk}(r[\mathrm{mid}], r[-1]; \mu_1)],\end{aligned}$$

where $[a; b]$ means the concatenating of the paths a and b. We regard the other samples as negative ones $\mathrm{neg}_r \in \mathcal{R} \setminus \{r, \mathrm{pos}_r\}$.

In addition to the above, hard negative samples sim_r are generated, distinguishing from normal negative samples neg_r. Specifically, the original node sequence r is used from the starting point to the middle point, and a random walk is used from the middle point to the end point:

$$\mathrm{sim}_r \leftarrow [r[0:\mathrm{mid}]; \mathrm{RandomWalk}(r[\mathrm{mid}], r[-1]; \mu_2)],$$

and the first half and the second half are switched and generated in the same way:

$$\mathrm{sim}_r \leftarrow [\mathrm{RandomWalk}(r[0], r[\mathrm{mid}]; \mu_2); r[\mathrm{mid}:-1]],$$

where the parameters are set to $\mu_1 > \mu_2$. Hereafter, r is sometimes denoted as org_r to indicate that it is the original node path.

3.2 Encoder

For an original sequence and data-augmented sequence r, we compute the representation vector of the nodes in the path. The representation vectors of nodes can be user demographic attributes or topics of posted texts for social graphs, or latitude and longitude or surrounding views for road networks. Alternatively, embedded vectors that take into account the graph structure, such as Word2Vec, GraRep, and LINE, can be used. The choice of vectorization method depends on what features of the node are taken into account. In our experiments, we adopt GraRep [11], which can represent adjacencies on the graph: $\mathbf{x}_u \leftarrow \mathrm{GraRep}(u; G)$. $\mathbf{x}_u$ is a vector that reflects the properties of node u in the graph G and tends to be a similar vector between neighboring nodes.

Next, a weighted average vector is computed using the weight $tanh()$ for the occurrence position ℓ_u in the node sequence to account for the occurrence position (See Fig.3):

$$\mathbf{y}_r \leftarrow \sum_{u \in r} \tanh\left(10\frac{\ell_u}{L_r} - 5\right) \mathbf{x}_u,$$

where $L_r = |r|$ denotes the path length. In a node sequence r, the nodes appearing in the early stages near the starting node s are weighted closer to -1, the Nodes appearing at the end of the sequence near the end node t are weighted closer to $+1$. This results in a feature vector of the opposite direction consisting of the same nodes having a Cosine similarity of -1.

3.3 Projection Head

The Projection head is applied to the feature vector $\mathbf{y}_r$ of a node sequence r, applying a nonlinear transformation. Specifically, it consists of a fully-connected layer with a bias term, a nonlinear transformation with tanh, and a fully-connected layer with a bias term:

$$\begin{aligned}\mathbf{z}_r &\leftarrow \mathbf{y}_r^T \mathbf{W}_{(\text{in})} + \mathbf{b}_{(\text{in})}\\ \mathbf{z}_r &\leftarrow \tanh(\mathbf{z}_r)\\ \mathbf{z}_r &\leftarrow \mathbf{z}_r^T \mathbf{W}_{(\text{out})} + \mathbf{b}_{(\text{out})}\end{aligned}$$

Previous studies on SimCLR have argued that having a projection head provides greater accuracy. In our experiment, we also compare the difference in results with and without the projection head.

3.4 NTXent Loss

The usual Normalized Temperature-Scaled Cross-Entropy loss employed in existing methods only distinguishes between positive and negative samples. On the other hand, the proposed method classifies node pairs into four types as shown in Fig 4: pos, sim, neg, and opp, and defines a loss function with different weights for each type. First, the cosine similarity between the embedding vectors of an original node sequence org_r and a node sequence pos_r with a common starting and ending point is denoted by $\cos(\text{org}_r, \text{pos}_r)$. Similarly, we define $\cos(\text{org}, \text{sim})$, $\cos(\text{org}, \text{neg})$, $\cos(\text{org}, \text{opp})$. Then, using the temperature parameters $\tau_1, \tau_2, \tau_3, \tau_4$ and the similarity control parameters $\lambda_1, \lambda_2, \lambda_3, \lambda_4$, we define a modified Normalized Temperature-Scaled Cross-Entropy for the node sequence r as follows:

$$\text{NTXent}(r) = \frac{P(\text{pos}; \tau_1)}{\lambda_1 P(\text{pos}; \tau_1) + \lambda_2 P(\text{sim}; \tau_2) + \lambda_3 P(\text{neg}; \tau_3) + \lambda_4 P(\text{opp}; \tau_4)}$$

where

$$\begin{aligned}P(\text{pos}; \tau_1) &= \exp\left(\frac{\cos(\text{org}, \text{pos})}{\tau_1}\right), \quad P(\text{sim}; \tau_2) = \exp\left(\frac{\cos(\text{org}, \text{sim})}{\tau_2}\right),\\ P(\text{neg}; \tau_3) &= \exp\left(\frac{\cos(\text{org}, \text{neg})}{\tau_3}\right), \quad P(\text{opp}; \tau_4) = \exp\left(\frac{\cos(\text{org}, \text{opp})}{\tau_4}\right).\end{aligned}$$

Then, we define a total loss for all node sequences as follows:

$$\sum_{r \in \mathcal{R}} \text{loss}(r) = -\sum_{r \in \mathcal{R}} \log(\text{NTXent}(r)).$$

By minimizing the loss function under the settings $\lambda_1 < \lambda_2 < \lambda_3 < \lambda_4$, we can obtain the embedded vector such that $\cos(\text{org}, \text{pos}) > \cos(\text{org}, \text{pos}) > \cos(\text{org}, \text{sim}) > \cos(\text{org}, \text{neg}) > \cos(\text{org}, \text{opp})$.

4 Experimental Design

4.1 Dataset

To evaluate the effectiveness of the proposed method, we use a road network, where the similarity between node sequences is easy to measure. The number

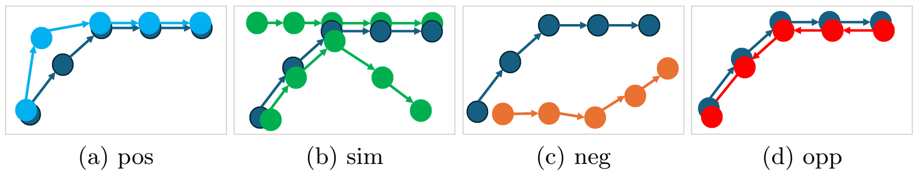


Fig. 4. Classification of node-sequence pairs

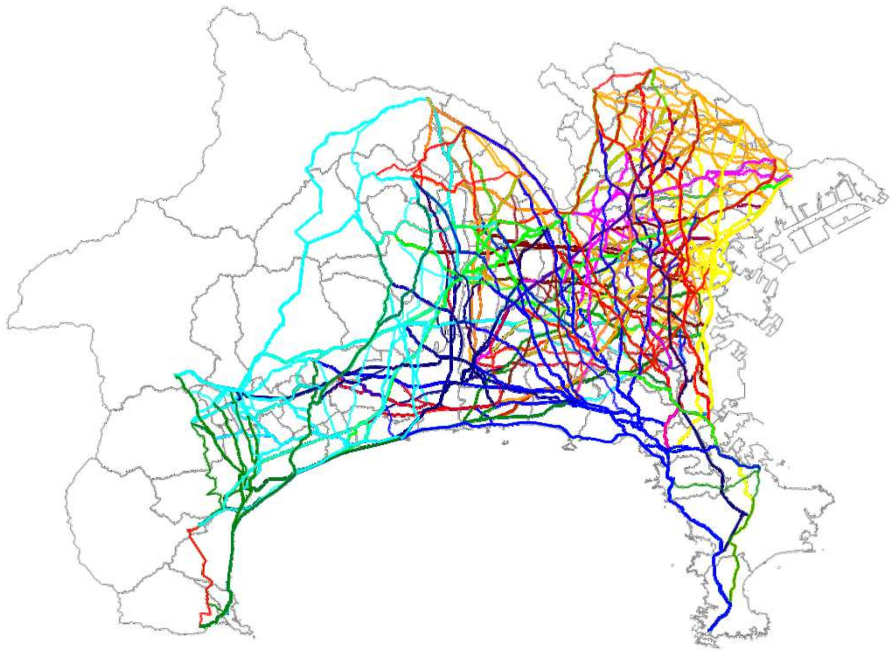


Fig. 5. Road network and shortest paths

of nodes is 295,151 and the number of edges is 402,576. For the node sequence, three locations were selected from 58 government offices in the target area, and a total of 185,136 shortest routes were generated as training data, connecting them as starting, transit, and ending points. Figure 5 shows the shortest paths between the 58 government offices for reference.

4.2 Settings

As shown in Fig.4, positive sample pos_r and hard negative sample sim_r are generated by data augmentation for the original node sequence org_r. In addition, the node sequence that is the exact opposite of the original is the reverse sample opp_r, and the others are the negative samples neg_r.

GraRep is employed as the Encoder layer of the proposed method, and the number of dimensions is set to 256. The dimension of the fully-connected layer with bias term in the Projection Head layer is set to 128. The similarity control parameters for modified NTXent loss were $\lambda_1 = 1, \lambda_2 = 2, \lambda_3 = 3, \lambda_4 = 4$ to learn a large pos similarity and a small opp similarity. The temperature parameters are set to $\tau_1 = \tau_2 = \tau_3 = \tau_4 = 0.5$, referring to the temperature parameters of SimCLR.

To train the proposed model, positive pos, hard-negative sim, negative neg, and opposite sample opp must be included in the same batch. Specifically, for a single node sequence org, one pos generated by data-augmentation with fixed start, transit, and end points, 55 sims generated with fixed start and transit points only, and 55 sims generated with fixed transit and end points. The batch also needs to include opp which exactly opposite to org, and its 1 pos and 110 sims. By treating node sequences that are neither pos, sim nor opp in the batch from each other as neg, four different node sequences can be included in a single batch. Therefore, the batch size was set to 224.

4.3 Comparison Methods

In this experiment, the following three embedding methods were used for comparison: the embedding vector $\mathbf{z}'_r$ learned by the usual NTXent as SimCLR:

$$\mathrm{Xent}(r) = \frac{\exp\left(\frac{\cos(\mathrm{org},\mathrm{pos})}{\tau}\right)}{\exp\left(\frac{\cos(\mathrm{org},\mathrm{pos})}{\tau}\right) + \sum_{\mathrm{neg}\in\mathcal{R}\setminus\{\mathrm{pos}\}} \exp\left(\frac{\cos(\mathrm{org},\mathrm{neg})}{\tau}\right)};$$

the output of the Encoder layer of the proposed method, $\mathbf{y}_r$, as Encoder; and the average vector of embedding vectors of nodes appearing in the node sequence r as Baseline:

$$\mathbf{g}_r \leftarrow \sum_{u\in r} \frac{1}{L_r} \mathbf{x}_u;$$

4.4 Evaluation Measure

In this experiment, we target a road network where it is easy to obtain similarity between node paths or between nodes. As the similarity between node paths, we use the average distance (Minimum Matching Distance) between a node u appearing in a sequence r_1 and the nearest neighbor node v in the other sequence r_2:

$$\mathrm{MMD}(r_1, r_2) = \frac{1}{2}\left\{\frac{1}{|r_1|}\sum_{u\in r_1}\min_{v\in r_2} \mathrm{euclid}(l_u, l_v)\right\} + \frac{1}{2}\left\{\frac{1}{|r_2|}\sum_{v\in r_2}\min_{u\in r_1} \mathrm{euclid}(l_v, l_u)\right\},$$

where the distance between nodes is calculated by the Euclidean distance $\mathrm{euclid}(l_u, l_v)$ of the coordinate vector l_u consisting of latitude and longitude.

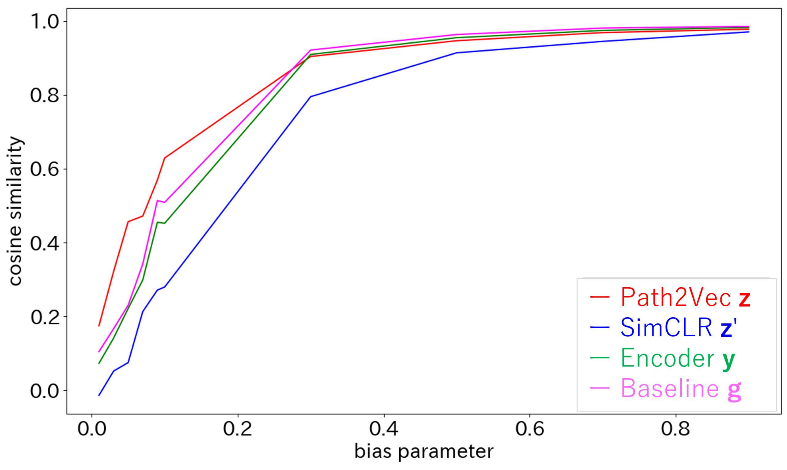


Fig. 6. Similarity between paths generated with different bias parameters

Then, the cosine similarity between the embedding vectors of the node sequences by the proposed method is obtained:

$$\mathrm{SIM}(\mathbf{z}_1, \mathbf{z}_2) = \frac{\langle \mathbf{z}_1, \mathbf{z}_2 \rangle}{\|\mathbf{z}_1\|\|\mathbf{z}_2\|}.$$

Similarly for the comparison method, the similarity between vectors is determined by the cosine similarity. The rank correlation coefficient between MMD and SIM is used as the evaluation index, since these similarities should show high values for the sequence of similar nodes.

5 Experimental Results

First, we check the similarity of the embedding vectors of the 10 node-sequences generated by changing the value of the random walk bias parameter to $\mu \in \{0.9, 0.7, 0.5, 0.3, 0.1, 0.09, 0.07, 0.05, 0.03, 0.01\}$ for each original node sequence org. Figure6 shows the bias parameter on the horizontal axis and the average cosine similarity on the vertical axis. Ideally, when the bias parameter is close to 1.0, the randomness of the random walk is lower and the similarity to org is higher. Conversely, as the parameter decreases, the randomness increases and the similarity to org decreases. Figure 6 shows that all four methods have an almost monotonically increasing trend, indicating ideal results.

Table 1 shows the similarity between the embedding vectors for the node sequences of the test data. Ideally, the node sequences should be less similar to each other from pos to sim, neg, and opp, in that order, with pos similarity close

Table 1. Similarity between embedding vectors of node sequences

Methods	pos	sim	neg	opp
Path2Vec $\mathbf{z}$	0.917	0.382	0.000	-0.921
SimCLR $\mathbf{z}'$	0.967	0.001	0.001	0.000
Encoder $\mathbf{y}$	0.839	0.355	0.000	-0.831
Baseline $\mathbf{g}$	0.784	0.670	0.434	0.822

Table 2. Rank correlation coefficient between SIM and MMD

Methods	pos	sim	neg	opp
Path2Vec $\mathbf{z}$	0.21	-0.02	-0.03	-0.20
SimCLR $\mathbf{z}'$	0.02	0.15	0.03	-0.18
Encoder $\mathbf{y}$	0.51	-0.06	0.00	-0.49
Baseline $\mathbf{g}$	-0.06	-0.07	0.16	-0.13

to $+1$, neg similarity close to 0, and opp similarity close to -1. Keeping the ideal result in mind, we can confirm that SimCLR and Baseline are not effective as embedding methods for node sequences, since the similarity of sim, neg, and opp from pos does not decrease in that order. The proposed methods, Path2Vec and Encoder, have an ideal order of similarity between node sequences. The similarity of pos is closer to $+1$, sim is closer to 0.5, neg is closer to 0, and opp is closer to -1 for Path2Vec than for Encoder. This confirms that the proposed method is the most effective embedding method for node sequences.

Table 2 shows the results of evaluating the rank correlation coefficients between the similarity SIM of embedding vectors and average minimum matching distance MMD for the test node sequences. Ideally, pos and sim should be positively correlated, neg uncorrelated, and opp negatively correlated. With the ideal result in mind, we can confirm that SimCLR and Baseline are not effective as node embedding methods because pos and sim are not positively correlated and opp is not negatively correlated. Path2Vec and Encoder are ideal except for the uncorrelation of sim, which confirms that the proposed method is effective as an embedding method for node sequences.

We compare the results of training the proposed method with different values of similarity control parameters for NTXent loss and temperature parameters in the proposed method. The parameter settings in the above experiment ($\lambda = (1,2,3,4), \tau = 0.5$) is param0, $\lambda = (4,3,2,1), \tau = 0.5$ is param1, $\lambda = (1,2,3,4), \tau = 0.05$ is param2.

Table 3 shows the similarity between the embedding vectors when trained with various parameter settings. The table 3 shows that for param2, similar to param0, the order of similarity is pos, sim, neg, and opp, but the differences between the classes are smaller. As for param1, the similarity is in the reverse order of the ideal, indicating that the desired embedding is difficult.

Table 3. Similarity between embedded vectors at each parameter setting value

Parameter sets	pos	sim	neg	opp
param0 ($\lambda = (1, 2, 3, 4), \tau = 0.5$)	0.917	0.382	0.000	−0.921
param1 ($\lambda = (4, 3, 2, 1), \tau = 0.5$)	−0.877	−0.233	0.000	0.811
param2 ($\lambda = (1, 2, 3, 4), \tau = 0.05$)	0.879	0.633	0.448	0.128

6 Conclusion

Considering the nature and appearance order of nodes in a node sequence, such as traffic routes on roads and information diffusion routes on SNS, we proposed a method to embed node sequence pairs with similar features in a graph in a vector space with high similarity and low similarity between node sequences without similar features. The technical novelties in the proposed method are 1) the ability to embed node sequences (paths) without teacher labels, 2) the ability to control the degree of similarity with the original sample to some extent by data augmentation based on random walk, and 3) the ability to express the diversity of similarity of node sequences by using modified NTXent loss. Evaluation experiments were conducted on real-world road network data, using the rank correlation coefficients between the average shortest matching distance and the similarity between vectors, as well as the relationship between the similarity of the embedded vectors. Experimental results showed that the proposed method is more effective than the compared methods as a node sequence embedding method because nodes with similar features and node sequences consisting of the same nodes in the graph can be represented as similar vectors, while nodes with different features and node sequences consisting of different nodes can be represented as dissimilar vectors.

There are three issues to be addressed in the future. The first is to test the model trained on the road network data used in this study on the road network data of other areas. The second issue is to verify whether the proposed method is effective for the followers relationships on social networking services and hyperlink relationships among web pages on the Internet. The third issue is to adjust the hyperparameters of the proposed method, such as similarity control, temperature, bias parameters, dimensionality of the node embedding method and training model, and the number of training epochs.

Acknowledgment

The first author is grateful for the financial support from JSPS Grant-in-Aid for Scientific Research (C) (No.22K12279).

References

1. C. Peng, W. Xiao, P. Jian, and W. Z., "A survey on network embedding," *CoRR*, vol. abs/1711.08752, 2017.

2. A. Grover and J. Leskovec, “Node2vec: Scalable feature learning for networks,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, (New York, NY, USA), pp. 855–864, Association for Computing Machinery, 2016.
3. H. J. Kim, S. E. Hong, and K. J. Cha, “seq2vec: Analyzing sequential data using multi-rank embedding vectors,” *Electronic Commerce Research and Applications*, vol. 43, p. 101003, 2020.
4. A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon, “A survey on contrastive self-supervised learning,” *CoRR*, vol. abs/2011.00362, 2020.
5. U. von Luxburg, “A tutorial on spectral clustering,” *Statistics and Computing*, vol. 17, pp. 395–416, Dec 2007.
6. T. M. J. Fruchterman and E. M. Reingold, “Graph drawing by force-directed placement.,” *Softw. Pract. Exp.*, vol. 21, no. 11, pp. 1129–1164, 1991.
7. Y. Hu, “Efficient and high quality force-directed graph drawing,” *Mathematica Journal*, vol. 10, pp. 37–71, 01 2005.
8. Y. Takeshi, S. Kazumi, and U. Naonori, “Cross-Entropy Directed Embedding of Network Data,” in *Proceedings of the 20th International Conference on Machine Learning*, pp. 832–839, AAAI Press, 2003.
9. B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: Online learning of social representations,” in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’14, (New York, NY, USA), pp. 701–710, Association for Computing Machinery, 2014.
10. J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, “Line: Large-scale information network embedding.,” in *Proceedings of the 24th International Conference on World Wide Web*, pp. 1067–1077, ACM, 2015.
11. S. Cao, W. Lu, and Q. Xu, “Grarep: Learning graph representations with global structural information,” in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM ’15, (New York, NY, USA), pp. 891–900, Association for Computing Machinery, 2015.
12. D. Wang, P. Cui, and W. Zhu, “Structural deep network embedding,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, (New York, NY, USA), pp. 1225–1234, Association for Computing Machinery, 2016.
13. C. Wang, C. Wang, Z. Wang, X. Ye, and P. S. Yu, “Edge2vec: Edge-based social network embedding,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 14, no. 4, pp. 1–24, 2020.
14. D. Zhou, J. Huang, and B. Schölkopf, “Learning with hypergraphs: Clustering, classification, and embedding,” in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS’06, (Cambridge, MA, USA), pp. 1601–1608, MIT Press, 2006.
15. S. Maleki, D. Wall, and K. Pingali, “Netvec: A scalable hypergraph embedding system,” *CoRR*, 03 2021.
16. D. Yang, B. Qu, J. Yang, and P. Cudre-Mauroux, “Revisiting user mobility and social relationships in lbsns: A hypergraph embedding approach,” in *The World Wide Web Conference*, WWW ’19, (New York, NY, USA), pp. 2147–2157, Association for Computing Machinery, 2019.
17. Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, “Hypergraph neural networks,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3558–3565, Jul. 2019.

18. Y. Gao, Y. Feng, S. Ji, and R. Ji, "Hgnn+: General hypergraph neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 3, pp. 3181–3199, 2023.
19. T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, pp. 1597–1607, PMLR, 2020.
20. I. Misra and v. d. L. Maaten, "Self-supervised learning of pretext-invariant representations," *CoRR*, vol. abs/1912.01991, 2019.
21. K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick, "Momentum contrast for unsupervised visual representation learning," *CoRR*, vol. abs/1911.05722, 2019.
22. M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," *CoRR*, vol. abs/2006.09882, 2020.
23. T. Wei, Y. You, T. Chen, Y. Shen, J. He, and Z. Wang, "Augmentations in hypergraph contrastive learning: fabricated and generative," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, (Red Hook, NY, USA), Curran Associates Inc., 2022.