Applied Network Science

RESEARCH

Open Access

Dynamic hypergraph embedding onto concentric hypersphere manifold intended for effective visualization

Shuta Ito[1] and Takayasu Fushimi[1*]

*Correspondence:
takayasu.fushimi@gmail.com

[1] School of Computer Science, Tokyo University of Technology, 1404-1 Katakura-machi, Hachioji-City, Tokyo 192–0982, Japan

Abstract

Hypergraph is a graph structure that can efficiently express the relationship of multiple nodes and has attracted attention in recent years. As with normal graphs, the structure changes every moment, and it is an important research topic in graph mining to capture structural changes. Many existing graph embedding methods focus on prediction tasks, and few focus on the visualization of structural changes. In this study, we aim to output embeddings for effective visualization in terms of spatial efficiency, node classification accuracy, graph structure maintenance, computational efficiency, and structural change detection performance. Our proposed method gets inspired by modularity maximization, quantifies connection strength between hypernodes and hyperedges, and embeds hypernodes on the surface of concentric spheres with a radius equal to the timestep, where a spherical surface has a wide area in the middle range. These devices are expected to correspond to the following two characteristics: (1) a graph has more node pairs whose distances are middle-range than short- and long-range; (2) a growing graph generally has an increasing number of nodes. Evaluation experiments using multiple real hypergraphs show that the proposed method is superior to existing visualization methods in the abovementioned terms.

Introduction

In graph theory, a graph represents usually a connection relation between two nodes, while hypergraphs can represent not only two nodes but also an arbitrary number of node connectivity relationships and have attracted attention because they are more expressive than general graphs (Feng et al. 2019; Do et al. 2020). To distinguish graphs that represent the connection relationship between two nodes from hypergraphs, we refer to them as normal graphs in the following. Since graph structure is difficult to numerically handle, many methods for obtaining vector representations of nodes and/or edges have been developed and using embedded vectors to solve link prediction, node classification, and clustering problems is becoming a standard methodology (Perozzi et al. 2014; Grover and Leskovec 2016; Tang et al. 2015; Cao et al. 2015). Therefore, methods for obtaining vector representations of nodes

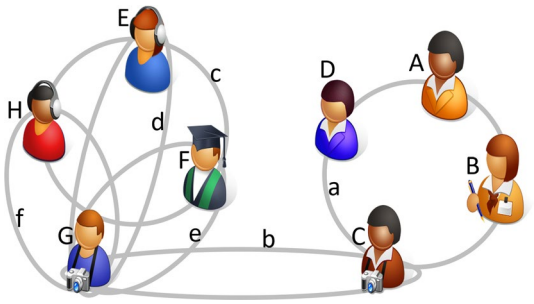


Fig. 1 Example of a co-author network 3: Persons with the uppercase character are (hyper) nodes, and gray lines with the lowercase character are (hyper) edges

and edges by embedding have become popular (Peng et al. 2017). Node embedding is a technique for converting a node to a low-dimensional vector while preserving its structural information. In particular, one method to uncovering the structure and inherent properties of complex real data is to embed it into a 2- or 3-dimensional vector and visualize it in Euclidean space. But, since the performance tends to degrade when directly acquiring 2- or 3- dimensional vectors by these embedding techniques, it is common to use t-SNE (Maaten and Hinton 2008). As embedding methods specialized in visualizing, although somewhat dated, Spring-Force embedding (Kamada and Kawai 1989) and Laplacian-Eigenmap embedding (Belkin and Niyogi 2003; von Luxburg 2007) exist. However, these embedding techniques and visualization methods cannot be directly applied to hypergraphs. Therefore, in this study, we propose an embedding method specialized in visualizing the representation of hypergraph structures.

In addition, graphs observed in the real world often change their structure. Changes in graph structure consist of the addition or deletion of nodes and edges or the rewiring of edges by combining them. Most existing studies on dynamic graphs focus on the pattern and frequency of structural changes, but few have focused on visualization of the impact of structural change. The addition or deletion of edges between different communities is considered to have a greater impact than that of edges within a single community. For example, a co-author network can be represented by a hypergraph as shown in Fig. 1.

In Fig. 1, hypernodes ABCD and EFGH are densely connected. Such a set of hypernodes or hyperedges is called a community. In many cases, co-authors are researchers in the same research field, and researchers in the same field are considered to be in a co-authoring relationship, so it can be regarded that a community is a research field. Suppose that hyperedge b is added later. This hyperedge represents a co-authoring relationship of different field researchers. Therefore, this is considered to be a large structural change, and the embedding vector is expected to drastically change.

The purpose of this study is to output embedding vectors for visualization that can detect important structural changes that undermine the community structure in dynamic hypergraphs. To this end, we propose a method that embeds the hypernodes of each snapshot in the dynamic hypergraph onto the spherical surface of concentric spheres. The features of the proposed method are shown below. The sphere surface is

suitable for visualizing many graph structures with a large number of middle-range node pairs because the near-range and far-range regions are narrow, and the middle-range region is wide. Since the radius is the time step, it is suitable for visualizing dynamic graphs in which the number of nodes increases with growth, and space-efficient embedding can be expected. In addition, since it is embedded in concentric spheres, it can be expected that it will be easy to detect large structural changes that affect the community structure. The proposed method quantifies the connection strength between hypernodes and hyperedges based on a matrix that plays a similar role to the modularity matrix in community extraction by double-centering the incidence matrix of the hypergraph, so a high accuracy of node classification can be expected. When obtaining the embedding vectors, it is calculated from the sparse incidence matrix instead of using the dense double-centered incidence matrix, so it is expected that the vector can be output at high speed with good computational efficiency. Our contribution in this study is listed below:

- We extend our existing hypergraph embedding method so as to apply it to dynamic hypergraphs.
- We reveal the relation between our method and the conventional community detection method based on modularity maximization.
- We quantitatively evaluate our method in terms of spatial efficiency, classification accuracy, graph structure maintenance, computational efficiency, and structural change detection performance.

The rest of this paper is organized as follows. "Related work" section reviews related work, and "Methodology" section explains our proposed method. "Experiments settings" section describes the experimental datasets and settings, and "Evaluation results" section reports and discusses the experimental results using real-world data. Finally, "Conclusion" section concludes and addresses future work.

Related work

This study proposes a novel embedding method of hypergraph intended for visualization. Our method uses a double-centered incidence matrix which is similar characteristics to the modularity matrix used for community detection. Therefore, we overview the existing studies in terms of embedding and visualization, clustering and community detection, and dynamic graph embedding.

Normal graph embedding

The embedding of nodes or graphs has been extensively studied over time. A classical approach involves utilizing the eigenvectors of the Laplacian matrix, which is called spectral clustering (von Luxburg 2007). This method proves effective for graphs with well-separated communities, as the Laplacian matrix is known for its clustering capabilities. Additionally, being a linear method, it can be executed relatively quickly. For arbitrary metric space objects, Laplacian Eigen Map constructs a neighborhood graph

such as a *k*-Nearest Neighbor graph on the distance between objects and performs low-dimensional embedding based on its Laplacian matrix (Belkin and Niyogi 2003).

There are specialized techniques for graph visualization, such as spring-force model (Fruchterman and Reingold 1991; Hu 2005) and cross-entropy method (Takeshi et al. 2003). The former aims to embed nodes in a way that matches the distance between them in the graph and their coordinates. The latter focuses on embedding pairs of nodes that are adjacent in the graph closer to each other. However, in recent years, alternative methods such as DeepWalk, Node2Vec, LINE, GraRep, and SDNE have emerged (Perozzi et al. 2014; Grover and Leskovec 2016; Cao et al. 2015; Tang et al. 2015; Wang et al. 2016). These methods focus on representing the structural features of nodes as vectors rather than visualizing graphs. For instance, DeepWalk and Node2Vec generate node sequences by conducting random walks starting from each node. They treat these sequences as sentences and apply Skip-Gram or CBoW to obtain node representation vectors. On the other hand, GraRep decomposes the adjacency matrix considering not only directly adjacent nodes but also nodes that are two or three hops away to obtain representation vectors.

Since these methods are designed for conventional graphs, some preprocessing steps, such as clique expansion to convert hypergraphs into normal graphs or star expansion to convert them into bipartite graphs, are necessary before applying them to the hypergraphs addressed in this paper. These methods are designed for the same or similar purpose of graph visualization or embedding, thus, in our experiments, we compare ours with some of these existing methods.

Hypergraph embedding

As an embedding method for hypergraphs, a method based on the Laplacian matrix of the clique-expanded graph has been proposed (Zhou et al. 2006). 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 Maleki et al. (2021), 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 (Yang et al. 2019). 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 (Maleki et al. 2021; Yang et al. 2019) do.

HGNN (Hypergraph Neural Networks) and its generalization, HGNN+, are neural network architectures designed to address node classification problems on hypergraphs (Feng et al. 2019; Gao et al. 2023). 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.

In this study, the objective is not to estimate node labels from node features and graph structure for classification problems. Instead, the goal is to acquire embeddings that enable effective visualization solely from the graph structure. Therefore, direct comparisons cannot be made.

Graph clustering and community detection

Graph clustering and community extraction have been studied for a long time, and many methods have been proposed. In community extraction, the concept/measure called modularity is important and has been adopted in many methods (Clauset et al. 2004; Blondel et al. 2008). Modularity is a measure showing how many edges in a cluster are larger than those in the case of a random graph based on the configuration model (Newman 2003). In other words, clustering that maximizes modularity is realized by deciding the cluster so that the number of edges closed in the cluster is relatively larger than the number of edges that span some clusters. However, in a large-scale graph, it is difficult to obtain an exact solution because the number of combinations is very large. Various approximation algorithms have been proposed to realize modularity maximization. Newman shows that the modularity can be expressed in terms of the eigenvectors of a characteristic matrix for the network, which is called the modularity matrix (Newman 2006).

Some studies generalize the modularity measure for normal graphs to apply to hypergraphs (Kumar et al. 2018; Kamiński et al. 2019). Kumar et al. (2018) calculated the modularity using the adjacency matrix in which the degree of each node was modified in response to the degree increasing due to the clique expansion. To achieve maximum modularity, the method repeats two phases: the phase of fast clustering of nodes based on the Louvain method (Blondel et al. 2008) and the phase of updating the weight for each hyperedge. The weight of the hyperedge that spans some clusters in a well-balanced

manner is reduced and updated so that it is easily cut in the next phase. The iteration is terminated when the updated amount of the weight matrix becomes sufficiently small, and the clustering result is output. Kamiński et al. (2019) generalized the Chung-Lu model and defined an exact modularity measure based on the model. To achieve clustering results that maximize modularity CNM (Clauset et al. 2004) like algorithm is employed.

In our study, we propose a method using a matrix similar to the modularity matrix, based on the idea that it is important to reflect the information of community structure in visualization. Unlike these studies, our method does not use the modularity matrix directly, but uses a double-centered incidence matrix, so we can realize a high-speed algorithm that takes advantage of the sparsity of the incidence matrix.

Dynamic graph embedding

Visualization of dynamic graphs has been attempted for a long time and is summarized in the literature (Beck et al. 2017). In Beck et al. (2017), the timelines are classified into animation, timeline, and hybrid, and the timelines are further classified into juxtaposed, superimposed, and integrated. Our method embeds hypernodes of each timestep into a concentric sphere, which allows us to know how much the angle from the origin has changed due to structural changes by embedding hypernodes at different time steps on spheres of different radii. Therefore, it can be said that our visualization method is classified as a timeline-juxtaposed visualization method.

Kang et al. propose a dynamic hypergraph neural network framework that focuses on key hyperedges (Kang et al. 2022). The authors introduce the concept of key hyperedges, which are selected based on their significance and impact on the overall hypergraph structure. These key hyperedges serve as critical elements for capturing and modeling the temporal dynamics of the hypergraph. The proposed framework leverages graph neural networks and attention mechanisms to effectively incorporate the information from key hyperedges and update node representations over time. By focusing on key hyperedges, the dynamic hypergraph neural network can adaptively learn the evolving patterns and dynamics in the hypergraph structure. This approach enables more accurate and informative analysis of dynamic hypergraphs in various applications, such as social networks, recommendation systems, and knowledge graphs. The experimental results demonstrate that the proposed method outperforms existing approaches in terms of both node classification accuracy and the ability to capture temporal dynamics in dynamic hypergraphs. Although Kang et al.'s method has excellent expressive power, it is not a method specialized for visualization, in fact, the authors did not present visualization results of dynamic hypergraphs.

Methodology

In this section, we formally explain the proposed method, its computational complexity and its relationship to the maximization of modularity.

Preliminary

First, we give the definition and notation. For a given hypergraph $H = (\mathcal{V}, \mathcal{E})$ which consists of sets of hypernodes $\mathcal{V}$ and hyperedges $\mathcal{E}$, we propose a method embedding

Table 1 Notation

Notation	Description
$H = (\mathcal{V}, \mathcal{E})$	Static hypergraph with sets of hypernodes $\mathcal{V}$ and hyperedges $\mathcal{E}$
$G = (\mathcal{V}, \mathcal{C})$	Normal graph with sets of normal-nodes $\mathcal{V}$ and normal-edges $\mathcal{C}$
$B = (\mathcal{V}, \mathcal{E}, \mathcal{R})$	Bipartite graph with sets of bipartite-nodes $\mathcal{V}, \mathcal{E}$ and bipartite-edges $\mathcal{R}$
$N = \lvert\mathcal{V}\rvert$	Number of hypernodes, number of normal-nodes
$M = \lvert\mathcal{E}\rvert$	Number of hyperedges
$L = \lvert\mathcal{R}\rvert$	Number of bipartite-edges
$K = \lvert\mathcal{C}\rvert$	Number of normal-edges
D	Number of dimensions of embedding vectors
$\mathbf{X} = [\mathbf{x}_v]_{v\in\mathcal{V}}$	Embedding vectors of hypernodes
$\mathbf{Y} = [\mathbf{y}_e]_{e\in\mathcal{E}}$	Embedding vectors of hyperedges
$\mathbf{H}$	Incidence matrix of a hypergraph, adjacency matrix of a bipartite graph
$\tilde{\mathbf{H}}$	Double-centered incidence matrices of a hypergraph
$\mathbf{A}$	Adjacency matrix of a normal graph
$\mathbf{I}_N$	$N \times N$ identity matrix
$\mathbf{J}_N$	$N \times N$ centering matrix
$\mathbf{1}_N$	N-dimensional vectors whose elements are all 1
$\mathcal{H}^{(T)}$	Dynamic hypergraph, sequence of T static hypergraphs
T	Number of hypergraph stnapshots

hypernodes $v \in \mathcal{V}$ and hyperedges $e \in \mathcal{E}$ into $(D-1)$-dimensional hypersphere manifold. Especially for the visualization, we set the dimension as $D = 3$, i.e., spherical surface. Here, since we express the number of dimensions of embedding vectors in Euclidean space as D, the number of dimensions of the manifold is $D-1$. Let $\mathbf{x}_v$ denote the embedding vector of hypernode v and $\mathbf{y}_e$ denote the embedding vector of hyperedge e. The incidence matrix of the hypergraph of $N = |\mathcal{V}|$ hypernodes and $M = |\mathcal{E}|$ hyperedges is represented as $N \times M$ matrix, $\mathbf{H}$, whose (v, e)-th element $h_{v,e}$ is 1 if hypernode v is included in hyperedge e, otherwise 0. Other notations are given in Table 1.

Revisit of static hypergraph embedding

Since the method proposed in this study is based on our existing one (Ito and Fushimi 2021), we revisit it. The method attempts to output the embedding vectors so that the connectivity among hypernodes and hyperedges is expressed as angles among vectors on a unit hypersphere. Formally, for the hypernodes and the hyperedge that are connected, $u, v, w \in e$, they are embedded in the same direction as viewed from the origin so that the cosine similarity among their embedding vectors $\mathbf{x}_u, \mathbf{x}_v, \mathbf{x}_w$ and $\mathbf{y}_e$ are large. That is, under the situation that the embedding vectors $\mathbf{x}$ of hypernodes are given, the embedding vector of hyperedge e that has the highest cosine similarity to the hypernodes $v \in e$ is found by

$$\mathbf{y}_e = \underset{\mathbf{y}\in\mathbb{R}^D, ||\mathbf{y}||=1}{\arg\max} \sum_{v\in e} \cos(\mathbf{y}, \mathbf{x}_v) \propto \sum_{v\in e} \mathbf{x}_v. \tag{1}$$

Similarly, given embedding vectors $\mathbf{y}$ of hyperedges, the embedding vector of hypernode v that has the highest cosine similarity to the hyperedges $\Gamma(v) = \{e \in \mathcal{E} | v \in e\}$ is obtained by

$$\mathbf{x}_v = \underset{\mathbf{x}\in\mathbb{R}^D, ||\mathbf{x}||=1}{\arg\max} \sum_{e\in\Gamma(v)} \cos(\mathbf{x}, \mathbf{y}_e) \propto \sum_{e\in\Gamma(v)} \mathbf{y}_e. \tag{2}$$

To represent the connectivity among N hypernodes and M hyperedges, the method focuses on the $N \times M$ incidence matrix $\mathbf{H}$. By using the value $h_{v,e}$, update formulae (1) and (2) turn to be $\mathbf{y}_e \propto \sum_{v\in\mathcal{V}} h_{v,e}\mathbf{x}_v$ and $\mathbf{x}_v \propto \sum_{e\in\mathcal{E}} h_{v,e}\mathbf{y}_e$.

Since matrix $\mathbf{H}$ only represents the connection or not by 0 or 1, to express the strength of connectivity by continuous scores ranging from -1 to $+1$, the method multiplies the centering matrices $\mathbf{J}_N = \mathbf{I}_N - \frac{1}{N}\mathbf{1}_N\mathbf{1}_N^T$ and $\mathbf{J}_M = \mathbf{I}_M - \frac{1}{M}\mathbf{1}_M\mathbf{1}_M^T$ from the left and the right, respectively, in the manner of the Young-Householder transform (Torgerson 1952):

$$\tilde{\mathbf{H}} = \mathbf{J}_N\mathbf{H}\mathbf{J}_M. \tag{3}$$

By using the value $\tilde{h}_{v,e}$ as a weight of connectivity strength between hypernode v and hyperedge e, the direction of the embedding vectors of more strongly connected hypernodes and hyperedges becomes more closely. As a result, update formulae (1) and (2) turn to be as follows:

$$\mathbf{y}_e \leftarrow \frac{\tilde{\mathbf{y}}_e}{||\tilde{\mathbf{y}}_e||}, \ \ \tilde{\mathbf{y}}_e \leftarrow \sum_{v\in\mathcal{V}} \tilde{h}_{v,e}\mathbf{x}_v, \tag{4}$$

$$\mathbf{x}_v \leftarrow \frac{\tilde{\mathbf{x}}_v}{||\tilde{\mathbf{x}}_v||}, \ \ \tilde{\mathbf{x}}_v \leftarrow \sum_{e\in\mathcal{E}} \tilde{h}_{v,e}\mathbf{y}_e. \tag{5}$$

These can be rewritten with a matrix–vector representation as follows:

$$\tilde{\mathbf{Y}} \leftarrow \tilde{\mathbf{H}}^T\mathbf{X} = \mathbf{J}_M\mathbf{H}^T\mathbf{J}_N\mathbf{X}, \ \ \tilde{\mathbf{X}} \leftarrow \tilde{\mathbf{H}}\mathbf{Y} = \mathbf{J}_N\mathbf{H}\mathbf{J}_M\mathbf{Y}. \tag{6}$$

Therefore, by multiplying the centered embedding vectors $\mathbf{J}_N\mathbf{X}$ and $\mathbf{J}_M\mathbf{Y}$ by a sparse matrix $\mathbf{H}$, and re-centered them by $\mathbf{J}_M$ and $\mathbf{J}_N$, we can obtain exactly the same results as multiplying a dense matrix $\tilde{\mathbf{H}}$.

Computational complexity

We show the whole algorithm of the method in Algorithm 1. Starting from the initialized vectors with random values, we repeat updating the vectors until convergence.

Algorithm 1 static hypergraph embedding: sHG_embed($H, D, \mathbf{X} = [-1, +1]^{N \times D}$)

Input: hypergraph $H = (\mathcal{V}, \mathcal{E})$, $N = |\mathcal{V}|$, $M = |\mathcal{E}|$
Input: number of dimensions D
Input: initial vectors $\mathbf{X}$ ▷ If not given, random values are set.
Output: embedding vectors $\mathbf{X} = [\mathbf{x}_v]_{v \in \mathcal{V}}$
Normalizing: $\forall v \in \mathcal{V}, \quad \mathbf{x}_v \leftarrow \frac{\mathbf{x}_v}{||\mathbf{x}_v||}$
while until convergence **do**
  $\bar{\mathbf{x}} \leftarrow \frac{1}{N} \sum_{v \in \mathcal{V}} \mathbf{x}_v$ ▷ centroid vector
  Centering: $\forall v \in \mathcal{V}, \quad \tilde{\mathbf{x}}_v \leftarrow \mathbf{x}_v - \bar{\mathbf{x}}$
  Merging: $\forall e \in \mathcal{E}, \quad \mathbf{y}_e \leftarrow \sum_{v \in e} \tilde{\mathbf{x}}_v$
  $\bar{\mathbf{y}} \leftarrow \frac{1}{M} \sum_{e \in \mathcal{E}} \mathbf{y}_e$ ▷ centroid vector
  Centering: $\forall e \in \mathcal{E}, \quad \tilde{\mathbf{y}}_e \leftarrow \mathbf{y}_e - \bar{\mathbf{y}}$
  Normalizing: $\forall e \in \mathcal{E}, \quad \mathbf{y}_e \leftarrow \frac{\tilde{\mathbf{y}}_e}{||\tilde{\mathbf{y}}_e||}$
  $\bar{\mathbf{y}} \leftarrow \frac{1}{M} \sum_{e \in \mathcal{E}} \mathbf{y}_e$ ▷ centroid vector
  Centering: $\forall e \in \mathcal{E}, \quad \tilde{\mathbf{y}}_e \leftarrow \mathbf{y}_e - \bar{\mathbf{y}}$
  Merging: $\forall v \in \mathcal{V}, \quad \mathbf{x}_v \leftarrow \sum_{e \in \Gamma(v)} \tilde{\mathbf{y}}_e$
  $\bar{\mathbf{x}} \leftarrow \frac{1}{N} \sum_{v \in \mathcal{V}} \mathbf{x}_v$ ▷ centroid vector
  Centering: $\forall v \in \mathcal{V}, \quad \tilde{\mathbf{x}}_v \leftarrow \mathbf{x}_v - \bar{\mathbf{x}}$
  Normalizing: $\forall v \in \mathcal{V}, \quad \mathbf{x}_v \leftarrow \frac{\tilde{\mathbf{x}}_v}{||\tilde{\mathbf{x}}_v||}$
end while

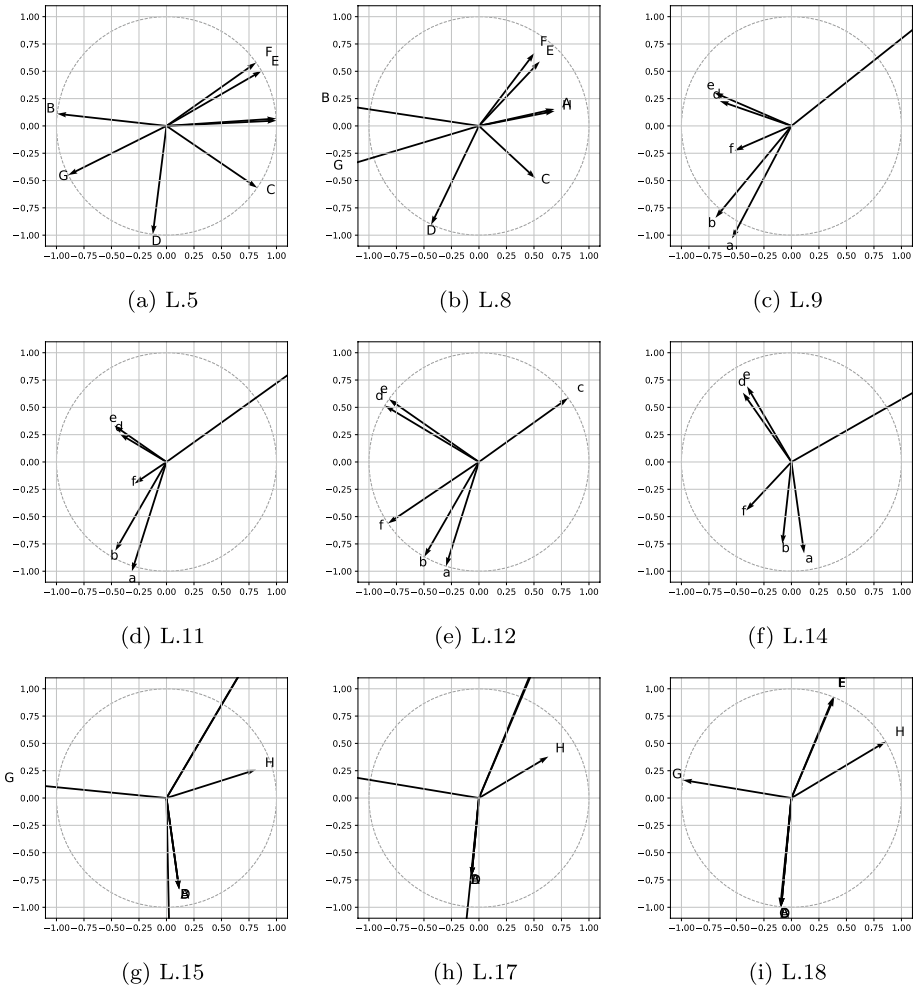
Figure 2 shows the embedding vectors when applying each step of the proposed algorithm to the sample graph in Fig. 1. In figures, uppercase and lowercase letters indicate hypernodes and hyperedges, respectively. The caption of each subfigure stands for the line number in Algorithm 1. First, hypernodes are embedded in a random manner in the unit circle (Fig. 2a), centering, merging, and normalizing are performed (Fig. 2b–h), at the end of the one iteration, nodes ABCD are overlapped, nodes EF are overlapped (Fig. 2i). Since these nodes are relatively strongly connected, they are embedded in the same direction as viewed from the origin. Conversely, nodes ABCD and nodes EFGH are embedded in opposite directions from the origin.

We consider the computational complexity of the method. As shown in Algorithm 1, the method takes $O(ND)$ to center the D-dimensional vectors of N hypernodes, $O(LD)$ to merge the D-dimensional vectors of $L = \sum_{e \in \mathcal{E}} |e|$ connected hypernodes, $O(MD)$ to center the D-dimensional vectors of M hyperedges, and $O(MD)$ to normalize them. A similar amount of time complexity is required when calculating the embedding vectors of a hypernodes. Therefore, the total complexity becomes $O(\alpha D(N + L + M))$, where α is the number of iterations until convergence. On the other hand, when we straightforwardly calculate the double-centered incidence matrix and use it, the total complexity gets to be $O(\alpha DNM)$ because $\tilde{\mathbf{H}}$ is full-matrix.

Relation to modularity maximization

The double-centered incidence matrix quantifies the relative strength of connectivity between hypernodes and hyperedges. Also, the modularity matrix proposed by Newman (2006) represents the difference between actual connections and those in the random null model. Here, we discuss the relation between these two matrices.

The (v, e)-th element of the double-centered incidence matrix can be written as



(a) L.5 (b) L.8 (c) L.9

(d) L.11 (e) L.12 (f) L.14

(g) L.15 (h) L.17 (i) L.18

Fig. 2 Example of updating vectors

$$\begin{aligned}\tilde{h}_{v,e} =& \left(h_{v,e} - \frac{1}{N}\sum_{u\in\mathcal{V}} h_{u,e}\right) - \frac{1}{M}\sum_{c\in\mathcal{E}}\left(h_{v,c} - \frac{1}{N}\sum_{u\in\mathcal{V}} h_{u,c}\right) \\ =& h_{v,e} - \left(\frac{|e|}{N} + \frac{|\Gamma(v)|}{M}\right) + \frac{L}{MN},\end{aligned} \tag{7}$$

where $1 \le |e| \le N$ and $1 \le |\Gamma(v)| \le M$ are the numbers of hypernodes connected to hyperedge e and hyperedges connected to hypernode v, respectively. Since the final term in Eq. (7) is constant independent from hypernodes and hyperedges, the value $\tilde{h}_{v,e}$ holds a larger value when a hypernode with a small degree is included in a small-size hyperedge, which rarely occurred, than when a hypernode with a large degree is included in a large-size hyperedge.

Similarly, we consider the modularity matrix. Let $B = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ be a bipartite graph obtained by applying the star-expansion to hypergraph $H = (\mathcal{V}, \mathcal{E})$, where $\mathcal{R} \subset \mathcal{V} \times \mathcal{E}$ is a set of bipartite edges. The adjacency matrix of the converted bipartite graph is equal to the incidence matrix of the hypergraph, so we use the same symbol $\mathbf{H}$.

According to the literature (Barber 2007), the (v, e)-th element of the modularity matrix $\mathbf{Q}$ for the bipartite graph is defined as

$$q_{v,e} = h_{v,e} - \frac{|\Gamma(v)| \cdot |e|}{L}, \tag{8}$$

where $\frac{|\Gamma(v)| \cdot |e|}{L}$ is the expected probability that bipartite-nodes v with degree $|\Gamma(v)|$ and e with degree $|e|$ are connected in the null model. Therefore, $q_{v,e}$ is a large value, when v and e are actually connected even though the expected connection probability is low. In this way, since our double-centered incidence matrix and the modularity matrix widely used in community detection are closely related, our method can be expected to produce the embedding vectors effective for clustering.

Next, we focus on the similarity of the algorithms. Now we define the matrices $\mathbf{\Phi} = \tilde{\mathbf{H}}\tilde{\mathbf{H}}^T$ and $\mathbf{\Psi} = \tilde{\mathbf{H}}^T\tilde{\mathbf{H}}$. Our method calculates the embedding vectors by iteratively multiplying matrices $\mathbf{X}$ and $\mathbf{Y}$ by the double-centered incidence matrices $\tilde{\mathbf{H}}^T$ and $\tilde{\mathbf{H}}$ as follows:

$$\begin{aligned}
\mathbf{Y}_1 &\leftarrow \tilde{\mathbf{H}}^T\mathbf{X}_0 \\
\mathbf{X}_1 &\leftarrow \tilde{\mathbf{H}}\mathbf{Y}_1 = \tilde{\mathbf{H}}\tilde{\mathbf{H}}^T\mathbf{X}_0 = \mathbf{\Phi}\mathbf{X}_0 \\
\mathbf{Y}_2 &\leftarrow \tilde{\mathbf{H}}^T\mathbf{X}_1 = \tilde{\mathbf{H}}^T\tilde{\mathbf{H}}\mathbf{Y}_1 = \mathbf{\Psi}\mathbf{Y}_1 \\
\mathbf{X}_2 &\leftarrow \tilde{\mathbf{H}}\mathbf{Y}_2 = \tilde{\mathbf{H}}\tilde{\mathbf{H}}^T\mathbf{X}_1 = \mathbf{\Phi}^2\mathbf{X}_0 \\
\mathbf{Y}_3 &\leftarrow \tilde{\mathbf{H}}^T\mathbf{X}_2 = \tilde{\mathbf{H}}^T\tilde{\mathbf{H}}\mathbf{Y}_2 = \mathbf{\Psi}^2\mathbf{Y}_1 \\
\mathbf{X}_s &\leftarrow \mathbf{\Phi}^s\mathbf{X}_0, \quad \mathbf{Y}_s \leftarrow \mathbf{\Psi}^{s-1}\mathbf{Y}_1,
\end{aligned} \tag{9}$$

where subscript s denotes the number of iteration steps. Thus, our algorithm is equal to conducting the power iteration for finding eigenvectors corresponding to the largest eigenvalue of $\mathbf{\Phi}$ and $\mathbf{\Psi}$, excluding the normalizing operation of vectors. Maximizing modularity also often uses spectral approaches such as spectral decomposition or singular value decomposition.

Dynamic hypergraph embedding

We extend the abovementioned embedding method for static hypergraphs so as to apply it to dynamic hypergraphs. We define a dynamic hypergraph $\mathcal{H}^{(T)} = (H^{(1)}, \ldots, H^{(T)})$, which is a sequence of T snapshots of a hypergraph, where $H^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$ denotes a snapshot of hypergraph at time step t. As shown in Algorithm 2, when inputting a dynamic hypergraph $\mathcal{H}^{(T)}$ and the number of dimensions of embedding vectors D, first the vectors of hypernodes $v \in \mathcal{V}^{(1)}$ are initialized as $\mathbf{x}_v^{(0)} \leftarrow [-1, +1]^D$ and normalized so as to whose norms are 1. Then, the algorithm calls function sHG_embed() with arguments $\mathbf{X}^{(0)}$ to obtain the vectors $\mathbf{X}^{(1)}$ of hypernodes $v \in \mathcal{V}^{(1)}$. In the same way, embedding vectors $\mathbf{X}^{(t-1)}$ are passed as the initial vector to the argument of function sHG_embed() to find the vector $\mathbf{X}^{(t)}$ according to Algorithm 1. The vectors of newly added hypernodes $v \in \mathcal{V}^{(t)} \setminus \bigcup_{t'<t} \mathcal{V}^{(t')}$ are initialized with random values. By setting the norm of embedding vectors of time step t as t, all snapshots are embedded into the concentric hypersphere. Our method that widens the embedding areas according to the time step naturally treats an evolving hypergraph whose number of hypernodes tends to increase.

Table 2 Basic statistics

Dataset	#classes	#HN	#HE	#BN	#BE	#NN	#NE
Coracoci	7	1330	1503	2833	9198	1330	8288
Coracoau	7	1676	723	2399	6926	1676	25,560
Citeseer	6	1019	819	1838	5616	1019	7734
Pubmed	3	3824	7951	11,775	69,210	3824	247,638

Algorithm 2 dynamic hypergraph embedding: dHG_embed($\mathcal{H}^{(T)}, D$)

Input: hypergraphs $\mathcal{H}^{(T)} = (H^{(1)}, \ldots, H^{(T)})$
Input: number of dimensions D
Output: embedding vectors $\mathcal{X} = (\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(T)})$
Initializing: $\mathbf{X}^{(0)} \leftarrow [-1, +1]^{N_1 \times D}$
Normalizing: $\forall v \in \mathcal{V}^{(1)}, \quad \mathbf{x}_v^{(0)} \leftarrow \frac{\mathbf{x}_v^{(0)}}{||\mathbf{x}_v^{(0)}||}$
for $1 \leq t \leq T$ **do**
    Updating: $\mathbf{X}^{(t)} \leftarrow t \cdot \text{sHG\_embed}(H^{(t)}, D, \mathbf{X}^{(t-1)})$
end for

In principle, the algorithm requires more iterations if the structures of $H^{(t)}$ and $H^{(t+1)}$ are significantly different, but converge in a few iterations if there are no significant changes.

Structural change extraction

In this section, we explain how to quantify the magnitude of structural change by focusing on the angle between embedding vectors. When there is a large structural change in the target hypergraph, it is desirable that the distance between the embedding vectors before and after the change is large. Conversely, it is desirable that the distance between the embedding vectors before and after a trivial structural change is small. Therefore, for each hypernode, we measure the circular distance (angle) between the vectors at time step t and $t+1$ to extract the magnitude of structural change:

$$\theta_v^{(t)} = \arccos \left\langle \frac{\mathbf{x}_v^{(t)}}{||\mathbf{x}_v^{(t)}||}, \frac{\mathbf{x}_v^{(t+1)}}{||\mathbf{x}_v^{(t+1)}||} \right\rangle. \tag{10}$$

The average angle of variation for all hypernodes is $\bar{\theta}^{(t)} = \frac{1}{N} \sum_{v \in \mathcal{V}} \theta_v^{(t)}$, and the structural changes with a large average angle of variation are extracted as important changes.

Experiments settings

Datasets

In our experimental evaluations, we employed the four real hypergraphs collected from LINQS,[1] and show their basic statistics in Table 2, where #HN, #HE, #BN, #BE, #NN, and #NE stand for the numbers of hypernode, hyperedge, bipartite-node, bipartite-edge,

[1] https://linqs.org/datasets/.

normal-node, and normal-edge, respectively. In each hypergraph, we extracted the maximum connected components. The Cora dataset consists of scientific papers classified into seven fields. We construct a citation hypergraph by taking each paper as a hyperedge and the set of papers cited by the paper as a hyperedge and extracting the maximum connected components of the hypergraph.

Compared methods

We explain the existing methods developed for visualizing and clustering normal graphs. Before applying the existing methods, we convert the target hypergraph to a normal graph by the clique expansion or a bipartite one by the star expansion.

For a given hypergraph $H = (\mathcal{V}, \mathcal{E})$, the clique expansion converts it to a normal graph $G = (\mathcal{V}, \mathcal{C})$ by connecting all the hypernodes in a hyperedge e by normal edges,

$$\mathcal{C} = \{(u, v) \in e \times e; e \in \mathcal{E} \wedge u \neq v\}.$$

Similarly, for a hypergraph $H = (\mathcal{V}, \mathcal{E})$, the star expansion converts it to a bipartite graph $B = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ by treating the hyperndoes $\mathcal{V}$ and hyperedges $\mathcal{E}$ as bipartite-nodes $\mathcal{U} = \mathcal{V} \cup \mathcal{E}$, and belonging relationships of hypernodes and hyperedges as bipartite-edges,

$$\mathcal{R} = \{(v, e) \in \mathcal{V} \times \mathcal{E}; v \in e\}.$$

Cross-entropy embedding

Cross-Entropy embedding (CE) calculates the vectors so as to minimize the binary cross entropy between the actual edge existence $a_{u,v}$ and the edge existence probability based on embedding vectors (Takeshi et al. 2003): $p_{u,v} = \exp\left(-\frac{1}{2}||\mathbf{x}_u - \mathbf{x}_v||^2\right)$. As for the normal graph, the loss function is defined as:

$$\begin{aligned}\mathrm{CE}(\mathbf{X}) &= -\sum_{(u,v)\in\mathcal{V}\times\mathcal{V}} a_{u,v}\log\left(p_{u,v}\right) - (1-a_{u,v})\log\left(1-p_{u,v}\right)\\ &= -\sum_{(u,v)\in\mathcal{C}} \log\left(p_{u,v}\right) - \sum_{(u,v)\in\bar{\mathcal{C}}} \log\left(1-p_{u,v}\right),\end{aligned}$$

and as for the bipartite graph, it is defined as:

$$\begin{aligned}\mathrm{CE}(\mathbf{X}) &= -\sum_{(u,v)\in\mathcal{U}\times\mathcal{U}} a_{u,v}\log\left(p_{u,v}\right) - (1-a_{u,v})\log\left(1-p_{u,v}\right)\\ &= -\sum_{(v,e)\in\mathcal{R}} \log\left(p_{v,e}\right) - \sum_{(v,e)\in\bar{\mathcal{R}}} \log\left(1-p_{v,e}\right)\\ &\quad -\sum_{(u,v)\in\mathcal{V}\times\mathcal{V}} \log\left(1-p_{u,v}\right) - \sum_{(c,e)\in\mathcal{E}\times\mathcal{E}} \log\left(1-p_{c,e}\right),\end{aligned}$$

where $\bar{\mathcal{C}} = (\mathcal{V}\times\mathcal{V})\backslash\mathcal{C}$ and $\bar{\mathcal{R}} = (\mathcal{V}\times\mathcal{E})\backslash\mathcal{R}$. Since these loss functions are used for link prediction tasks, we can expect to obtain the embedding vectors where connectivities are well considered. We optimize the loss function via the quasi-Newton method, and the computational complexity is $O(\beta N^2 D)$ for a normal graph and $O(\beta(N+M)^2 D)$ for

a bipartite graph because the method needs to consider the connectivity of all the node pairs. Here β is the number of iterations, and the value tends to be very large.

Spring-force embedding

Spring-Force embedding (SF) calculates the vectors so as to minimize the difference between the graph distance $g_{u,v}$ and the Euclidean distance based on embedding vectors (Kamada and Kawai 1989): $d_{u,v} = ||\mathbf{x}_u - \mathbf{x}_v||$. As for the normal graph, the loss function is defined as:

$$\mathrm{SF}(\mathbf{X}) = \sum_{(\mathrm{u,v})\in\mathcal{V}\times\mathcal{V}} \kappa_{\mathrm{u,v}}(\mathrm{g_{u,v}} - \mathrm{d_{u,v}})^2,$$

and as for the bipartite graph, it is defined as:

$$\mathrm{SF}(\mathbf{X}) = \sum_{(\mathrm{u,v})\in\mathcal{U}\times\mathcal{U}} \kappa_{\mathrm{u,v}}(\mathrm{g_{u,v}} - \mathrm{d_{u,v}})^2,$$

where $\kappa_{u,v}$ is a spring constant which is normally set to $1/2g_{u,v}^2$. Here, we emphasize that the graph distance of $u, v \in \mathcal{V}$ in a bipartite graph is twice that in a normal graph, so almost the same results are to be obtained. We optimize the loss function via the quasi-Newton method, and the computational complexity is $O(\gamma N^2 D)$ for a normal graph and $O(\gamma(N+M)^2 D)$ for a bipartite graph because the method needs to consider the distance of all the node pairs. Here γ is the number of iterations, and the value tends to be large but not so large as that of CE, β.

Laplacian-eigenmaps embedding

Laplacian-Eigenmaps embedding (LE) also known as Spectral embedding calculates the vectors so as to minimize the distance between the embedding vectors of adjacent node pairs (Chung 1997). As for the normal graph and bipartite graph, the loss function is defined as:

$$\mathrm{LE}(\mathbf{X}) = \sum_{(\mathrm{u,v})\in\mathcal{V}\times\mathcal{V}} \mathrm{a_{u,v}}||\mathbf{x}_\mathrm{u} - \mathbf{x}_\mathrm{v}||^2,$$

and as for the bipartite graph, it is defined as:

$$\mathrm{LE}(\mathbf{X}) = \sum_{(\mathrm{u,v})\in\mathcal{U}\times\mathcal{U}} \mathrm{a_{u,v}}||\mathbf{x}_\mathrm{u} - \mathbf{x}_\mathrm{v}||^2.$$

Since this minimization problem can be solved as an eigenvalue problem, we find the eigenvectors via the power-iteration method. The computational complexity is $O(\delta KD)$ for a normal graph and $O(\delta LD)$ for a bipartite graph because the method only needs to consider the degrees of adjacent node pairs. Here δ is the number of iterations, and the value tends to be small.

DeepWalk

DeepWalk (DW) is a graph embedding technique that learns representations for nodes in a graph by treating random walks as sentences and applying word2vec-like algorithms

to learn node embeddings (Perozzi et al. 2014). Specifically, it performs multiple random walks and records the order in which nodes are visited. Using the sequences obtained from these random walks, techniques like Skip-gram or Negative Sampling are employed to learn the node embeddings. These embeddings capture relationships and similarities between nodes, allowing for the representation of graph structure in a lower-dimensional space. In this study, we set the values of parameters as follows according to the default settings:

- Dimensions: 64
- Walk length: 40
- Number of walks: 10
- Workers: 1
- Seed: 1
- Window size: 5

Node2Vec

Node2Vec (NV) is a slight modification of DeepWalk's random walk method (Grover and Leskovec 2016). It introduces two parameters, p and q, where p determines the probability of returning to a previously visited node during a random walk, and q determines the probability of exploring an unvisited node. The parameter p adjusts the probability of finding a small fraction of the node's local neighborhood, while q adjusts the probability of finding a larger fraction of the node's global neighborhood. The remaining steps of the embedding process are the same as in DW. In this study, we set the values of parameters as follows according to the default settings:

- Dimensions: 128
- Walk length: 50
- Number of walks: 5
- p and q: 1
- Workers: 1
- Seed: 1
- Window size: 10
- Batch words: 4

GraRep

GraRep (GR) is a graph embedding technique that captures higher-order proximity information by decomposing the adjacency matrix into a series of matrices (Cao et al. 2015). It factorizes these matrices to obtain node embeddings in a low-dimensional space, incorporating both local and global structural information. The resulting embeddings capture complex patterns and dependencies, improving performance in tasks like node classification and link prediction. In this study, we set the values of parameters as follows according to the default settings:

- Dimensions: 16
- Number of adjacency matrix powers: 5
- Seed: 1
- Number of iterations: 20

After obtaining node embeddings using DeepWalk (DW), Node2Vec (NV), and GraRep (GR), we performed T-distributed Stochastic Neighbor Embedding (t-SNE) with the following settings: n_components=2, perplexity=30, n_iter=1000.

Evaluation results

In this section, we show the evaluation results of our proposed method (HM: Hypersphere-Manifold embedding) and six existing methods (CE, SF, LE, DW, NV, and GR). We apply HM to original hypergraphs and the existing methods to normal graphs and bipartite graphs. As for the bipartite graph, we only show the results of hypernode embeddings without those of hyperedges.

Visualization of static hypergraph

First, we show the visualization results in Figs. 3, 4, 5, 6 and 7. Here, the embedding vector of the existing method is adjusted so that it fits in the range of 0 to 1. In each figure, only hypernodes are plotted, and their colors are based on their classes shown in Table 2. Also, the title of each figure describes the name of embedding and expansion methods, e.g., CE-Star means the results of the cross-entropy embedding for the bipartite graph obtained by the star expansion.

From Figs. 3, 4, 5 and 6, the followings can be observed. CE-Cliq and CE-Star output rich visualization, where nodes are scattered in space and nodes with different colors are well-separated. The number of nodes in CE-Cliq looks somewhat smaller than that of CE-Star, this is because the clique expansion converts a hyperedge to a clique where all nodes are structurally equivalent and the CE embedding outputs the same vectors for structural equivalent nodes. SF-Cliq and SF-Star output almost the same results, where nodes are distributed in concentric circles according to the distance from the latent root node. LE-Cliq and LE-Star output poor visualization, where many nodes are overlapped. DW-* and NV-* output similar results with each other; and output poor results such as different colored nodes located nearby. GR-Cliq and GR-Star output good results in terms of classification because different colored nodes are relatively separated, but many nodes are embedded into nearby positions, so spatial efficiency seems to be bad. From Fig. 7, HM outputs rich visualization, where nodes are scattered in space and nodes with different colors are well-separated as well as CE. In what follows, we quantitatively compare them in terms of spatial efficiency, classification accuracy, correlation to the graph distance, and execution time.

Spatial efficiency

In visualization, if the nodes overlap each other, it becomes difficult to visually grasp the relationship between the nodes. Therefore, it is necessary to be able to represent the graph using the full space. In order to quantify whether the nodes are embedded all over the space, the spatial efficiency is measured by the ratio of the random points that have at least

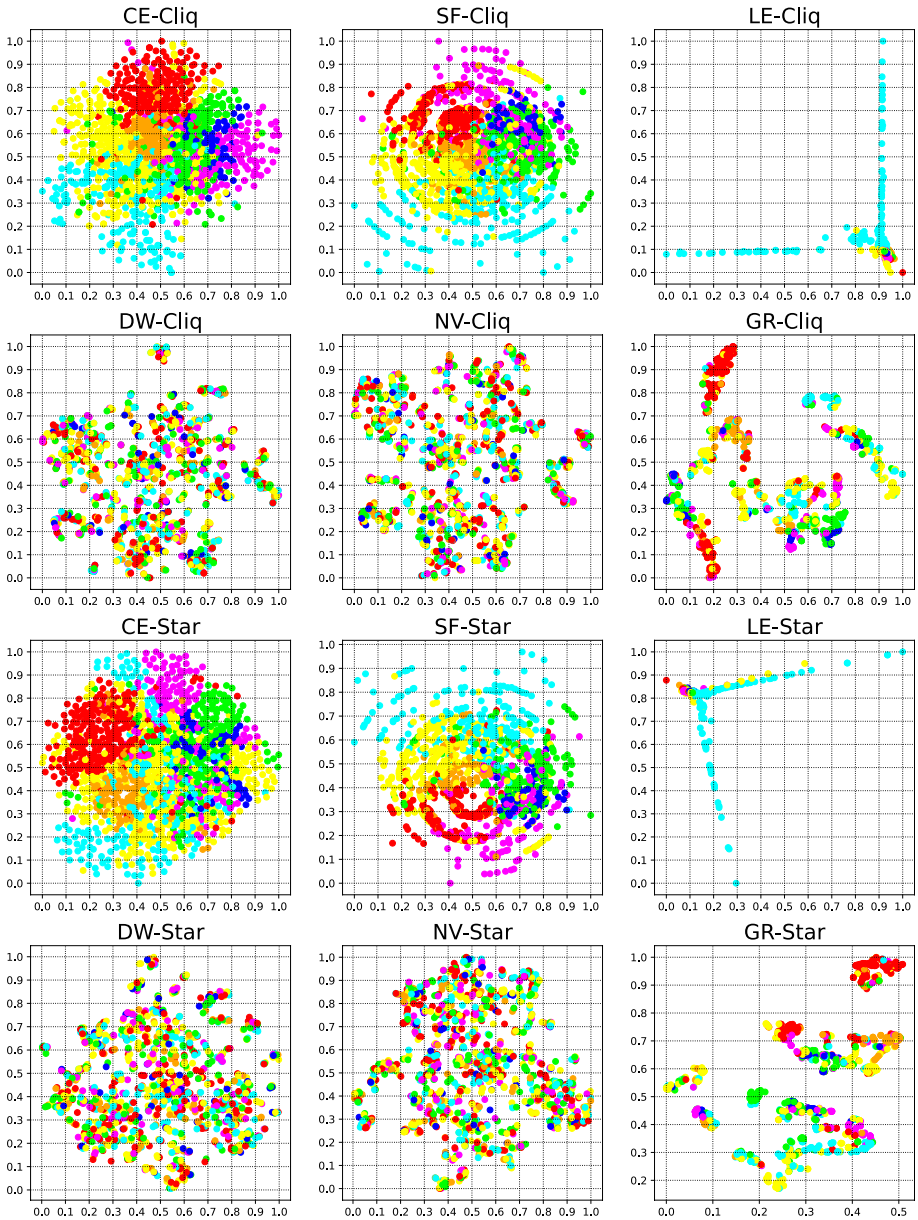


Fig. 3 Visualization results by existing methods (coracoci)

one node within the radius of r. We generated 100 uniform random points $o \in \mathcal{O}$, and set the radius based on the minimum Euclidean distance for existing methods:

$$r = \frac{1}{100} \sum_{o \in \mathcal{O}} \min_{o' \in \mathcal{O}, o' \neq o} \sqrt{2 - 2\langle \mathbf{x}_o, \mathbf{x}_{o'} \rangle},$$

and based on the minimum circular distance for our method:

$$r = \frac{1}{100} \sum_{o \in \mathcal{O}} \min_{o' \in \mathcal{O}, o' \neq o} \arccos \langle \mathbf{x}_o, \mathbf{x}_{o'} \rangle.$$

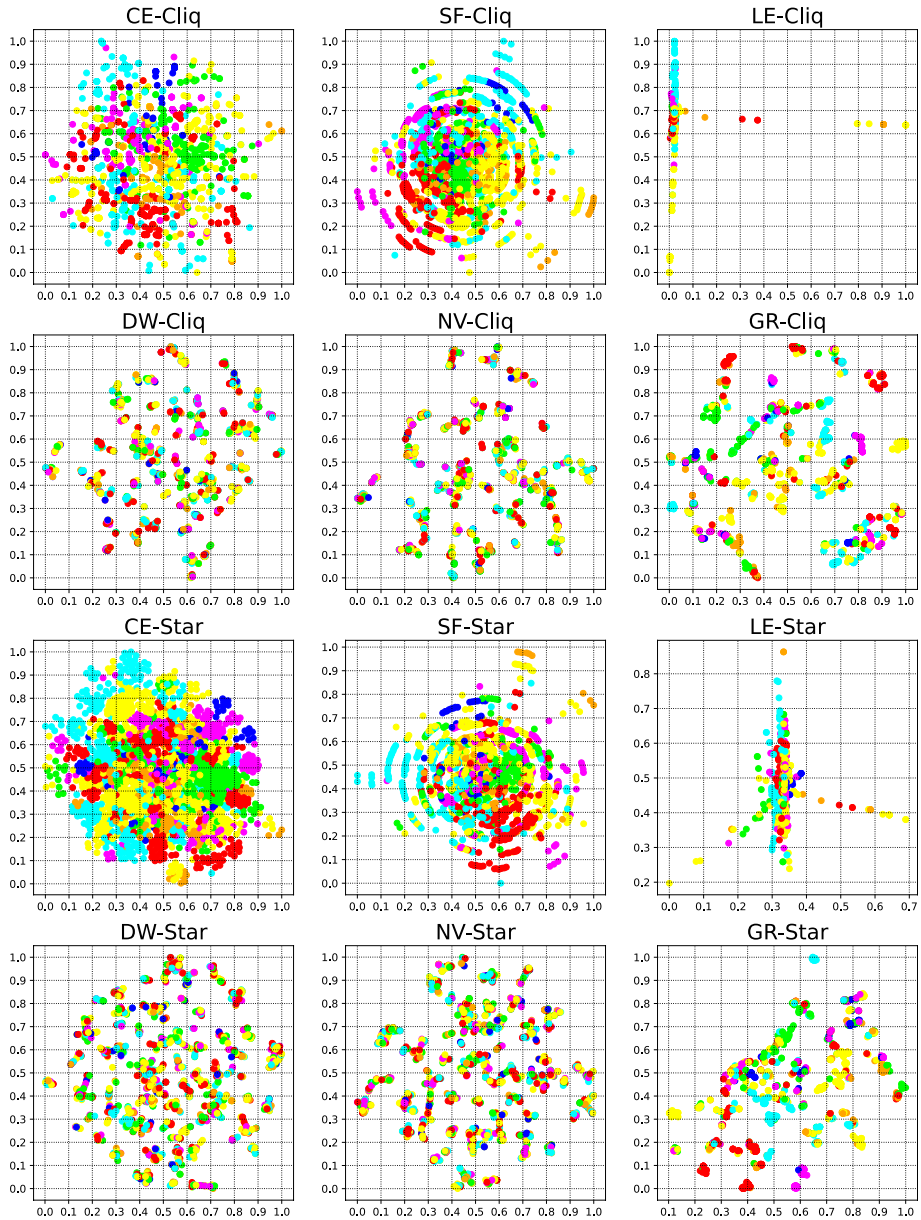


Fig. 4 Visualization results by existing methods (coracoau)

Table 3 shows the spatial efficiency, where the bold one is the best and italic one is the second best. From Table 3, as seen in the visualization results in the previous section, CE and SF, which are specialized for graph visualization, and HM, which is the proposed method, can embed all nodes with spatial efficiency. Although it is not as good as CE and SF, DW and NV output the visualization result with good spatial efficiency. On the other hand, linear methods such as LE cannot fully use the space. GR gave visualization results with poor spatial efficiency, though not as bad as LE.

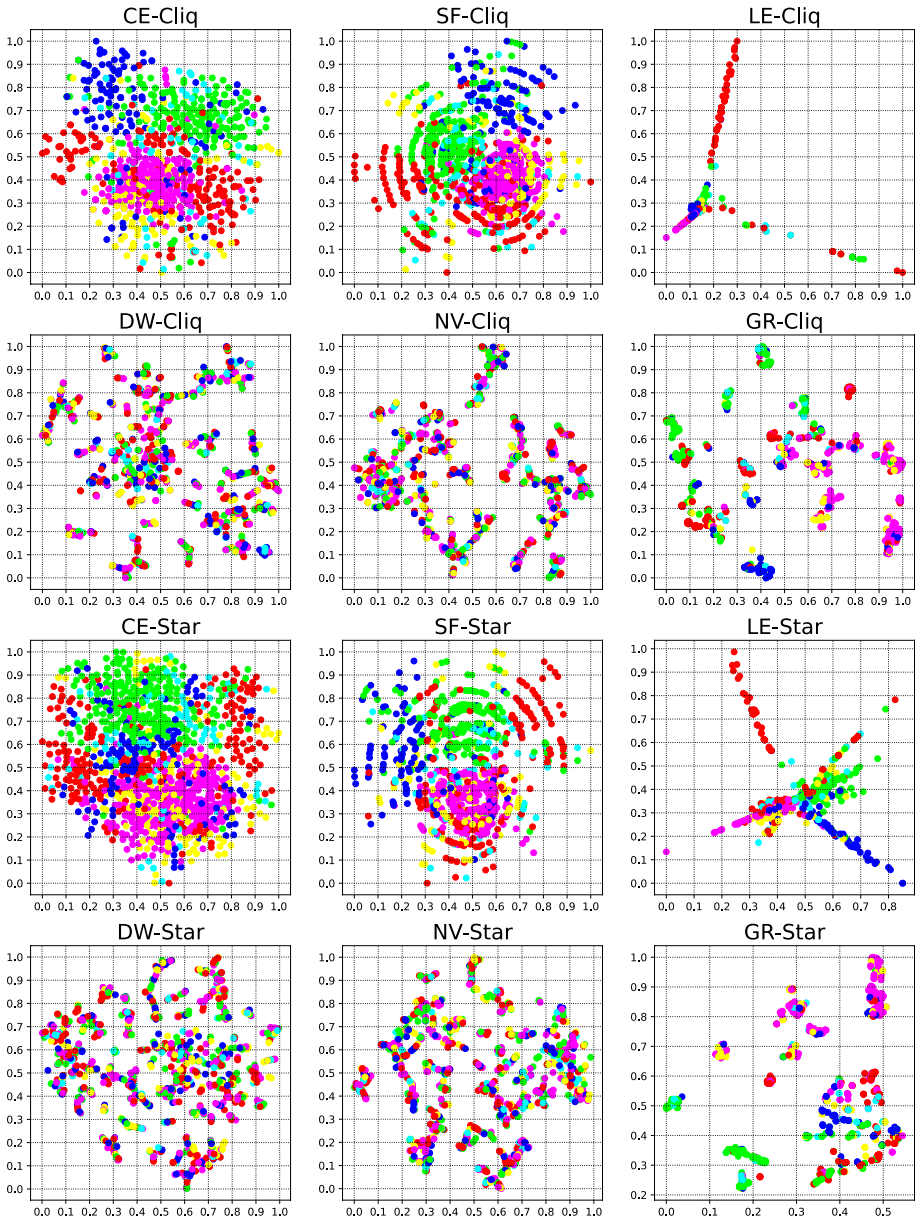


Fig. 5 Visualization results by existing methods (citeseer)

Classification accuracy

The community structure is one of the characteristics of graphs, and how effectively the community structure can be expressed in visualization is an important point of view. In other words, it is an essential condition that nodes with the same label are located nearby. In this study, we performed a classification task for the embedding vectors, and evaluated the extent to which nodes with the same label are located in the neighborhood by classification accuracy. In order to quantify the classification accuracy, we employed Support Vector Machine (SVM), Random Forests (RF), and Light Gradient Boosting Machine (LGBM) as a classifier, and use the embedding

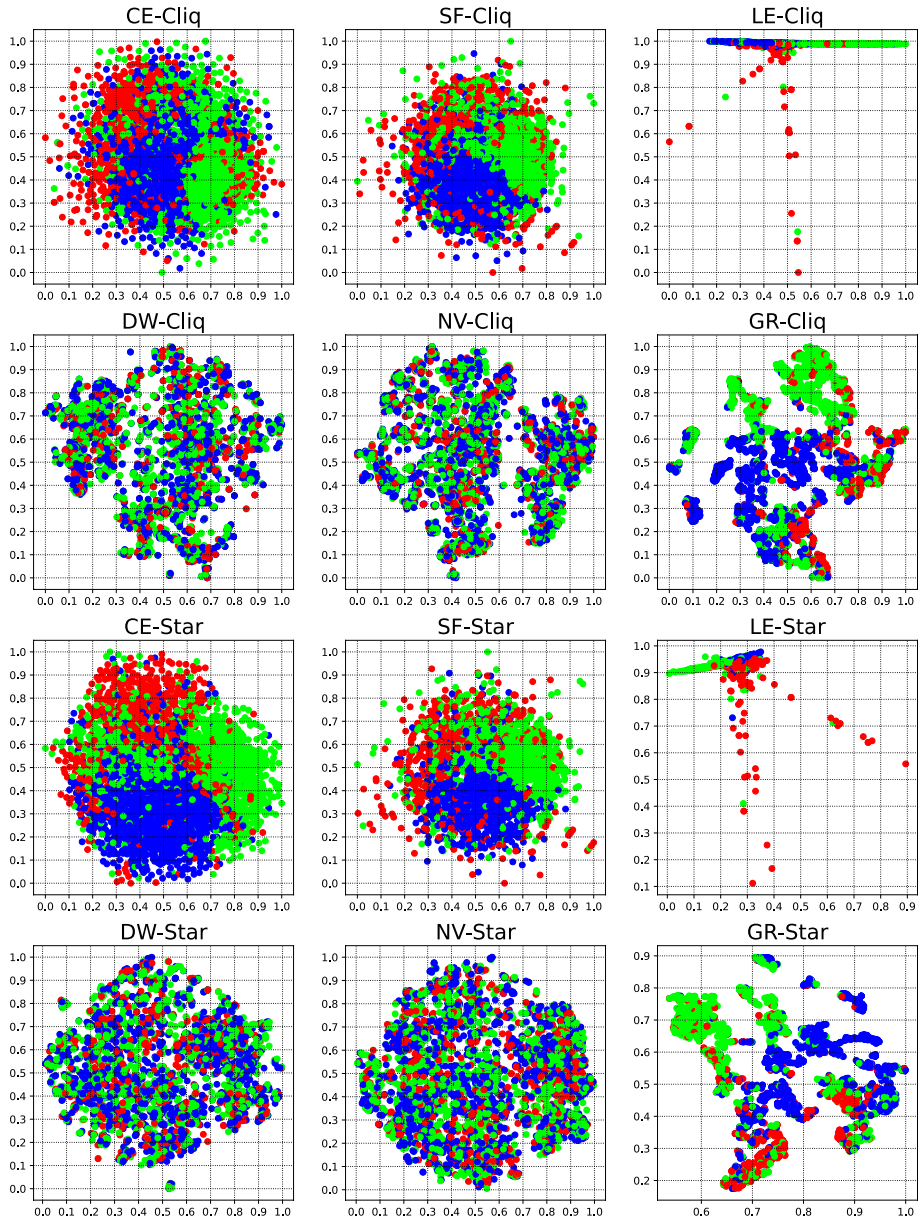
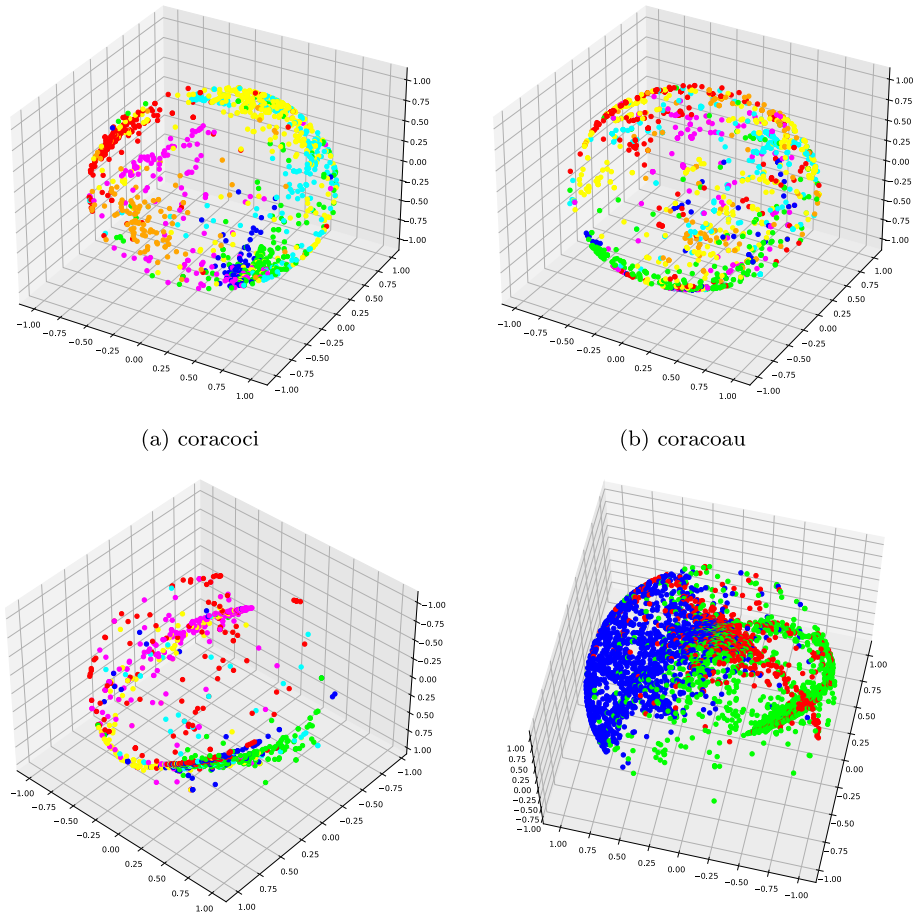


Fig. 6 Visualization results by existing methods (pubmed)

vector as a feature. In SVM, we used the polynomial kernel (SVM-POL) and radial basis function kernel (SVM-RBF). We conducted 5-fold cross-validation and the accuracy is shown in Table 4.

From Table 4, we observed the following. HM outperforms the existing methods although not in all the datasets. Especially in the case of SVM-POL, the difference from the second best is large in coracoci and pubmed datasets. It was found that CE and SF not only achieve efficient spatial visualization like HM but also demonstrate high classification accuracy. In contrast, LE and GR exhibit high classification accuracy despite having lower spatial efficiency, in other words, it was quantitatively confirmed that nodes with the same label exhibited overlap. In addition, while not all, CE-Cliq achieves higher



(a) coracoci (b) coracoau

(c) citeseer (d) pubmed

Fig. 7 Visualization results by proposed method

Table 3 Spatial efficiency

	Coracoci	Coracoau	Citeseer	Pubmed
HM	**0.90**	**0.95**	0.67	*0.80*
CE-Cliq	0.72	0.71	*0.71*	**0.83**
CE-Star	*0.77*	*0.74*	**0.83**	0.75
SF-Cliq	0.75	0.59	0.64	0.75
SF-Star	0.75	0.59	0.64	0.75
LE-Cliq	0.17	0.08	0.15	0.15
LE-Star	0.17	0.25	0.26	0.27
DW-Cliq	0.61	0.54	0.64	0.74
DW-Star	0.65	0.64	0.66	0.70
NV-Cliq	0.69	0.47	0.51	0.69
NV-Star	0.64	0.55	0.53	0.76
GR-Cliq	0.45	0.53	0.39	0.61
GR-Star	0.32	0.39	0.35	0.46

Table 4 Classification accuracy

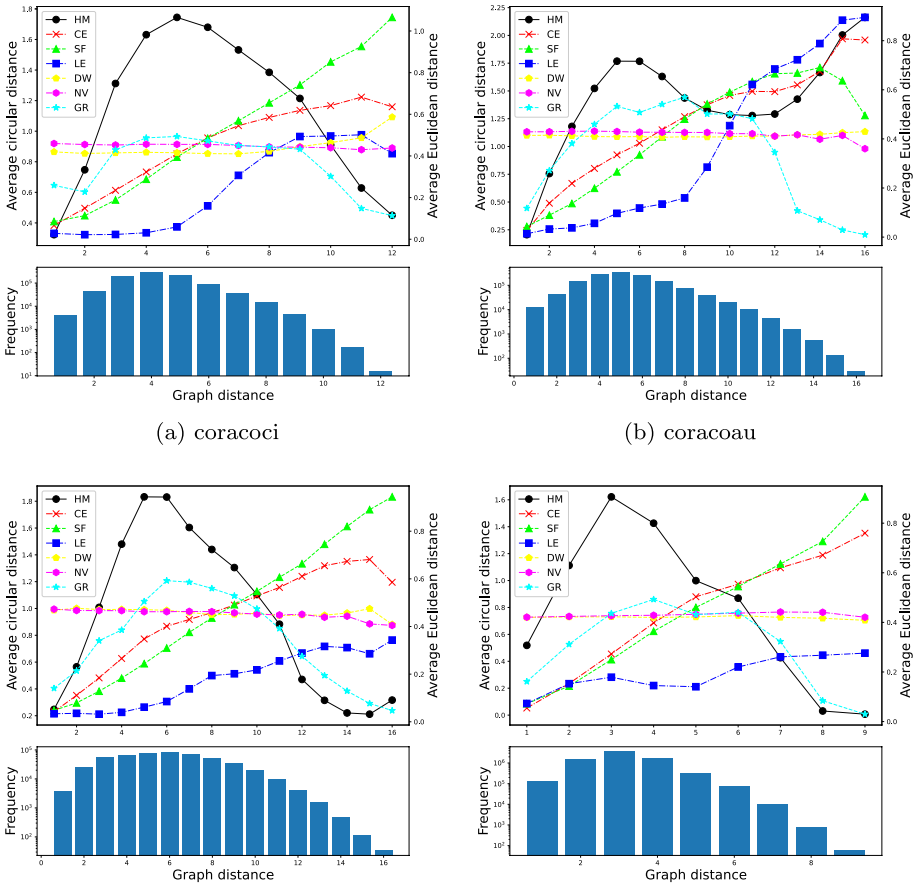
	Coracoci	Coracoau	Citeseer	Pubmed	Coracoci	Coracoau	Citeseer	Pubmed
	SVM-POL				SVM-RBF			
HM	**0.66**	**0.48**	*0.52*	**0.74**	**0.65**	**0.53**	0.53	**0.74**
CE-Cliq	*0.50*	*0.46*	**0.54**	0.59	*0.64*	0.44	**0.64**	0.72
CE-Star	0.43	0.32	0.46	*0.68*	0.56	0.45	0.55	**0.74**
SF-Cliq	0.45	0.38	**0.54**	0.64	0.61	0.49	*0.63*	*0.73*
SF-Star	0.45	0.34	0.49	*0.68*	0.61	0.48	0.58	**0.74**
LE-Cliq	0.33	0.30	0.31	0.57	0.39	0.36	0.56	0.68
LE-Star	0.32	0.29	0.50	0.63	0.34	0.38	0.60	*0.73*
DW-Cliq	0.26	0.27	0.25	0.40	0.26	0.27	0.24	0.39
DW-Star	0.26	0.27	0.25	0.41	0.26	0.27	0.23	0.41
NV-Cliq	0.26	0.27	0.25	0.41	0.26	0.27	0.23	0.40
NV-Star	0.26	0.27	0.25	0.41	0.26	0.27	0.26	0.40
GR-Cliq	0.46	0.40	0.51	0.65	0.54	*0.52*	0.57	*0.73*
GR-Star	0.41	0.41	0.5	0.59	0.53	*0.52*	0.57	0.72
	RF				LGBM			
HM	**0.67**	*0.57*	0.56	**0.76**	*0.66*	*0.68*	0.56	*0.74*
CE-Cliq	*0.66*	**0.58**	**0.66**	0.73	**0.67**	**0.71**	**0.65**	*0.74*
CE-Star	0.56	0.48	0.57	0.74	0.58	0.61	0.56	**0.75**
SF-Cliq	0.60	0.51	*0.62*	0.73	0.55	0.53	0.56	0.72
SF-Star	0.59	0.48	0.58	*0.75*	0.56	0.55	0.54	0.72
LE-Cliq	0.48	0.51	0.56	0.71	0.51	0.66	0.55	0.68
LE-Star	0.49	0.51	*0.62*	0.73	0.55	0.64	*0.60*	0.71
DW-Cliq	0.27	0.26	0.22	0.39	0.18	0.17	0.18	0.38
DW-Star	0.26	0.27	0.24	0.40	0.17	0.18	0.20	0.38
NV-Cliq	0.25	0.26	0.20	0.40	0.16	0.15	0.16	0.39
NV-Star	0.26	0.26	0.25	0.41	0.18	0.18	0.20	0.40
GR-Cliq	0.56	0.56	0.61	0.73	*0.66*	0.67	0.58	**0.75**
GR-Star	0.58	0.56	0.60	0.72	0.63	0.67	0.59	*0.74*

accuracy compared to CE-Star, and GR-Cliq performs better than GR-Star in terms of accuracy. DW and GR did not achieve high accuracy in any of the classifiers.

Correlation to graph distance

In order to quantify the maintenance degree of the graph structure, the distance between embedding vectors for all node pairs was calculated, and the average value was plotted for each graph distance. In the upper row of Fig. 8, the average of the circular distance for HM is represented by a black line and the averages of the Euclidean distances for existing methods are represented by colored lines. The lower row of Fig. 8 depicts the distribution of the graph distance. Here, since the distance between hypernodes in the bipartite graph is twice the distance in a normal graph and the results are very similar, we omitted the results of *-Star.

From Fig. 8, the following observations can be seen. In SF, the average distance increases linearly according to the graph distance, that is, indicating a strong correlation. CE is as not as SF, but is correlated with the graph distance. LE also shows a weak correlation. HM has a strong correlation in short distances but has no correlation in long distances. On the other hand, looking at the distribution of the graph distance, we can see



(a) coracoci (b) coracoau

(c) citeseer (d) pubmed

Fig. 8 Correlation to graph distance

that there are many medium-distance node pairs, and there are few short- or long-distance node pairs. In other words, it can be said that it does not have a significant effect on seeing the graph structure even if there is no correlation between a node pair that is too far away. Furthermore, on a sphere, the space around each node is narrow, the space farthest from each node is also narrow, and the space in the center is the largest, which is consistent with the distance distribution of the graph, making it suitable for graph plotting. In DW and NV, there is no correlation between the graph distance and the distance between embeddings, and it can be said that the graph structure cannot be reflected. GR tends to be similar to HM. This is probably because GR considers both local and global proximity information.

Execution time

Table 5 shows the execution time of each method after expansions. From Table 5, it can be said that CE, which can produce rich visualization, needs much computational load; LE, which is a linear method, needs less computational load; HM is superior to existing methods in that it can output results of the same quality as CE and at the same speed as LE. In addition, *-Star takes more time than *-Cliq, this is because the number of nodes in a bipartite graph obtained by the star expansion is larger than that in a normal graph

Table 5 Execution time (s)

	Coracoci	Coracoau	Citeseer	Pubmed
HM	*1.22*	4.99	1.19	**2.04**
CE-Cliq	131.75	329.62	89.94	606.93
CE-Star	2273.03	1293.17	660.86	19989.00
SF-Cliq	8.63	28.78	6.36	91.85
SF-Star	30.20	27.07	17.84	409.00
LE-Cliq	**0.38**	*2.86*	*1.10*	11.80
LE-Star	1.64	**0.36**	**0.14**	*7.91*
DW-Cliq	17.03	22.67	12.33	52.98
DW-Star	37.28	29.40	23.71	183.05
NV-Cliq	15.37	19.89	10.54	135.91
NV-Star	33.33	26.03	19.59	159.00
GR-Cliq	4.87	5.62	2.69	44.57
GR-Star	5.32	3.80	3.37	52.44

by the clique expansion. However, although not included in execution time, the execution time for the clique expansion is generally larger than that for the star expansion because of the number of edges.

Visualization of dynamic hypergraph

Next, we evaluate the ability to embed dynamic hypergraphs. Figure 9 depicts visualization results of a dynamic hypergraph $\mathcal{H}^{(3)} = (H^{(1)}, H^{(2)}, H^{(3)})$ artificially synthesized, where nodes represented by circles are $\mathcal{V}^{(1)}$, crosses are $\mathcal{V}^{(2)}$, triangles are $\mathcal{V}^{(3)}$, and edges represented by dotted lines connect the same nodes. We set the original of coracoci as $H^{(1)}$, construct $H^{(2)}$ by adding rewirings for 1% bipartite-edge of $H^{(1)}$, and $H^{(3)}$ by adding further rewirings for 1% bipartite-edges of $H^{(2)}$. Edge-rewiring is according to the configuration model (Newman 2003). From Fig. 9, we can see that HM embeds the nodes of each timestep as vectors of corresponding radii in concentric spheres so that we can distinguish the nodes of different timesteps and detect some structural changes; CE, SF, and GR produce rich visualization results even for the dynamic hypergraphs but it isn't easy to distinguish a node-set from those of different timesteps. In LE, DW, and NV, the same node is placed in completely different places depending on the timestep. Although the random number seed is fixed and the vector of the previous time step is taken into consideration, the embeddings of the nodes that are not directly related to the edge-rewiring have also changed significantly. Although not shown, the results of *-Star for other hypergraphs have a similar tendency to those of *-Cliq. Figure 10 presents the visualization results obtained by the effective embedding methods, HM, CE, SF, and GR, at different timesteps.

Structural change extraction

To quantitatively evaluate the extraction ability of structural changes in a dynamic hypergraph, we measure the distance of embedding vectors before and after adding structural changes. Structural changes are artificially added according to the configuration model as same as in the previous section. We generated 100 rewired hypergraphs with certain rewiring probability $p \in \{0.1, 0.2, \ldots, 0.9\}$.

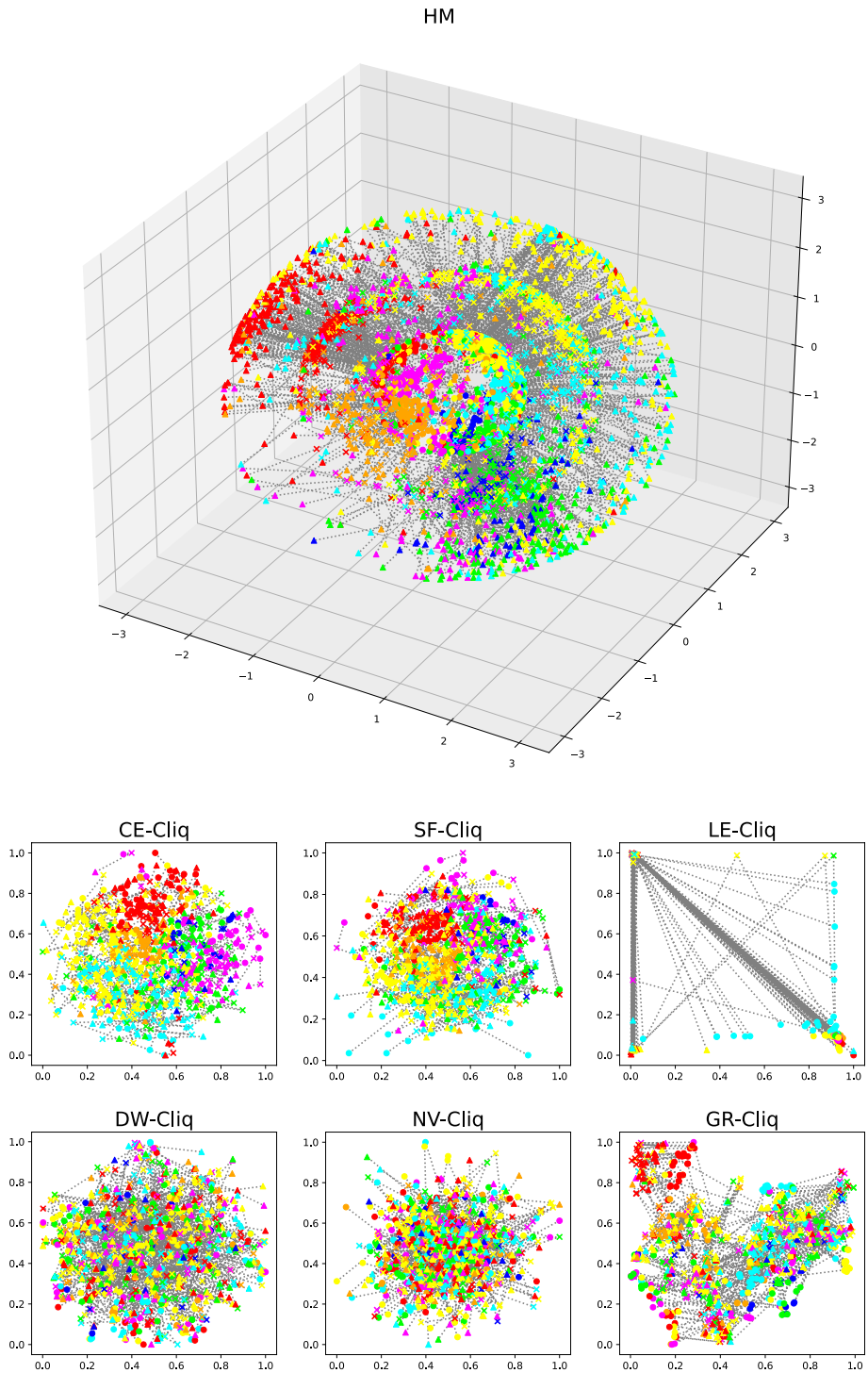
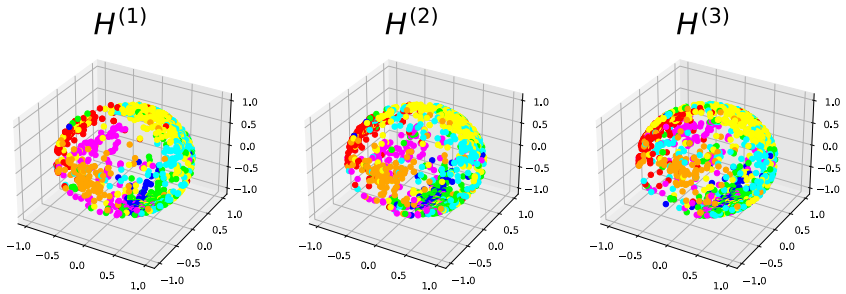
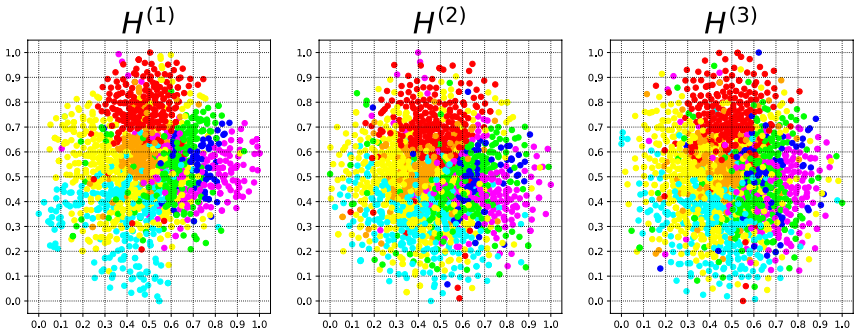


Fig. 9 Visualization of dynamic hypergraph (coracoci)

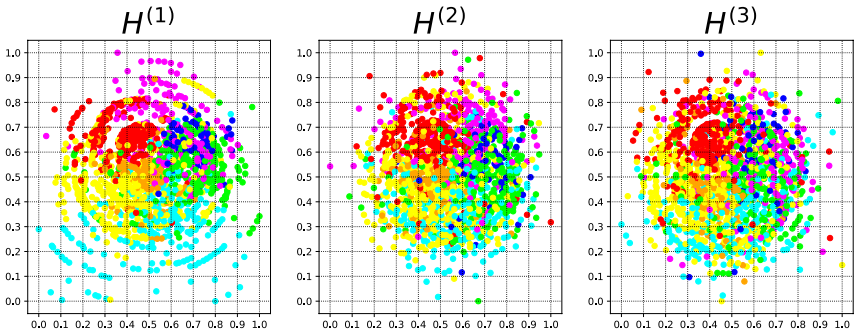
Figure 11 shows the average circular distance for HM and the average Euclidean distance for the existing methods, where the horizontal axis is the rewiring probability. Due to the complication of the figure, only the results of *-Cliq are shown, but similar results were obtained with *-Star. The distances are normalized by being divided by the



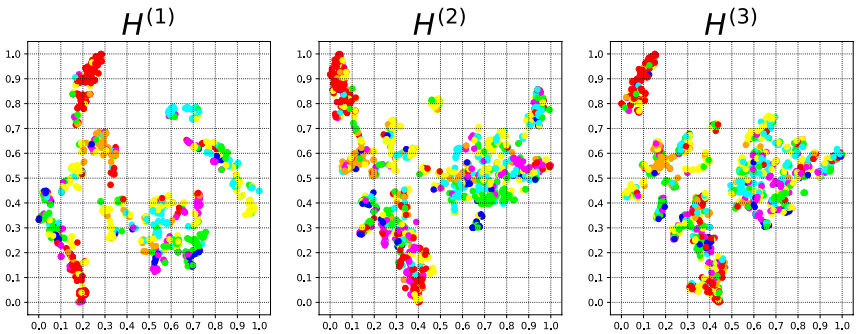
(a) Hypersphere-Manifold



(b) Cross-Entropy



(c) Spring-Force



(d) GraRep

Fig. 10 Visualization of snapshots of dynamic hypergraph (coracoci)

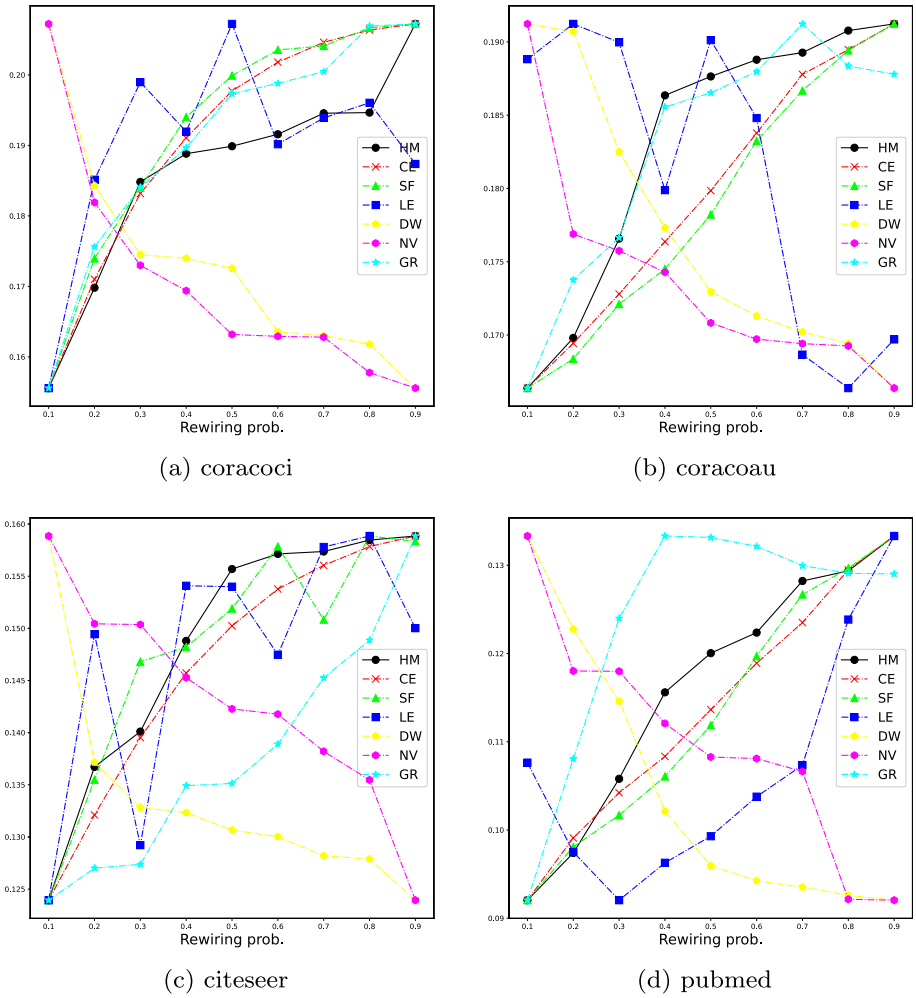


Fig. 11 Distance between before and after structural changes

theoretical maximum distance π and $\sqrt{2}$. From Fig. 11, we can see that the average distance by HM, CE, SF, and GR increases in proportion to the rewiring probability, that is the embedding vectors by HM, CE, and SF moved in proportion to the scale of structural changes. In other words, it can be seen that the embedding vector of each node does not change much with a small structural change, while the embedding vector of each node moves greatly with a large structural change that affects the community structure. In LE, the vector of nodes changes regardless of the magnitude of the change. Two random walk-based methods (DW and NV) showed that the larger the structural change, the smaller the change of the node vector.

Conclusion

In this study, we proposed a dynamic hypergraph embedding method based on our existing method of embedding static hypergraphs. We formally redefined the base method of our method and clarified its relationship with community extraction based on modularity maximization. In the base method, a static hypergraph is embedded in the hypersphere. For visualization, it is a method of embedding on a 2-dimensional manifold in

3-dimensional Euclidean space, that is, on a sphere. Similar to a normal graph, a hypergraph has a small number of nodes at short and long distances from each node, but has many nodes at intermediate distances, so it can be embedded on a sphere to achieve visualization with good spatial efficiency. Furthermore, since the objective function and solving algorithm are similar to those of modularity maximization, embedding with high node classification accuracy can be realized. An extended method embeds the hypergraph of each timestep on the sphere of concentric spheres. The proposed method, which uses the timestep as the radius, enlarges the drawing area toward the outside, so it is suitable for drawing a dynamic hypergraph that grows with the number of nodes increasing with the passage of time. In addition, since the algorithm does not explicitly multiply the double-centered incidence matrix and the embedding vectors, the results can be obtained quickly.

To confirm the usefulness of our proposed method, we compared it with existing embedding methods by using four real hypergraphs. As a result, it was confirmed that the spatial efficiency is good, the classification accuracy is high, and the embedding vector that can easily capture structural changes can be output at high speed.

In future work, we plan to confirm the applicability to a wider variety of hypergraphs with different structures and to consider an efficient visualization method for hyperedges.

Acknowledgements
Not applicable.

Author contributions
SI performed the research and wrote the article. TF contributed to designing the proposed method and part of experimental evaluations. All authors read and approved the final manuscript.

Funding
The first author is grateful for the financial support by JSPS KAKENHI Grant Number JP22K12279.

Availability of data and materials
The Julia, Python codes and raw datasets used and analysed during the current study will be available at https://github.com/fuppo27/HHME.git.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
All authors declare no financial and non-financial competing interests.

Received: 12 March 2023 Accepted: 27 June 2023
Published online: 07 July 2023

References

Barber M (2007) Modularity and community detection in bipartite networks. Phys Rev E 76:066102. https://doi.org/10.1103/PhysRevE.76.066102
Beck F, Burch M, Diehl S, Weiskopf D (2017) A taxonomy and survey of dynamic graph visualization. Comput Graph Forum 36(1):133–159. https://doi.org/10.1111/cgf.12791
Belkin M, Niyogi P (2003) Laplacian eigenmaps for dimensionality reduction and data representation. Neural Comput 15(6):1373–1396
Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. J Stat Mech Theory Exp 2008(10):P10008

Cao S, Lu W, Xu Q (2015) Grarep: learning graph representations with global structural information. In: Proceedings of the 24th ACM international on conference on information and knowledge management, CIKM '15. Association for Computing Machinery, New York, NY, USA, pp 891–900. https://doi.org/10.1145/2806416.2806512
Chung FRK (1997) Spectral graph theory. American Mathematical Society, New York
Clauset A, Newman MEJ, Moore C (2004) Finding community structure in very large networks. Phys Rev E. https://doi.org/10.1103/physreve.70.066111
Do M, Yoon S, Hooi B, Shin K (2020) Structural patterns and generative models of real-world hypergraphs. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, KDD 20. Association for Computing Machinery, New York, NY, USA, pp 176–186. https://doi.org/10.1145/3394486.3403060
Feng Y, You H, Zhang Z, Ji R, Gao Y (2019) Hypergraph neural networks. In: Proceedings of the AAAI conference on artificial intelligence, vol 33(01), pp 3558–3565. https://doi.org/10.1609/aaai.v33i01.33013558. https://ojs.aaai.org/index.php/AAAI/article/view/4235
Fruchterman TMJ, Reingold EM (1991) Graph drawing by force-directed placement. Softw Pract Exp 21(11):1129–1164
Gao Y, Feng Y, Ji S, Ji R (2023) Hgnn+: general hypergraph neural networks. IEEE Trans Pattern Anal Mach Intell 45(3):3181–3199. https://doi.org/10.1109/TPAMI.2022.3182052
Grover A, Leskovec J (2016) Node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, KDD '16. Association for Computing Machinery, New York, NY, USA, pp 855–864. https://doi.org/10.1145/2939672.2939754
Hu Y (2005) Efficient and high quality force-directed graph drawing. Math J 10:37–71
Ito S, Fushimi T (2021) High-speed and noise-robust embedding of hypergraphs based on double-centered incidence matrix. In: Benito RM, Cherifi C, Cherifi H, Moro E, Rocha LM, Sales-Pardo M (eds) Complex Networks & Their Applications X. Springer, Cham, pp 536–548
Kamada T, Kawai S (1989) An algorithm for drawing general undirected graphs. Inf Process Lett 31:7–15
Kamiński B, Poulin V, Pralat P, Szufel P, Théberge F (2019) Clustering via hypergraph modularity. PLOS ONE 14(11):1–15. https://doi.org/10.1371/journal.pone.0224307
Kang X, Li X, Yao H, Li D, Jiang B, Peng X, Wu T, Qi S, Dong L (2022) Dynamic hypergraph neural networks based on key hyperedges. Inf Sci 616:37–51. https://doi.org/10.1016/j.ins.2022.10.006
Kumar T, Vaidyanathan S, Ananthapadmanabhan H, Parthasarathy S, Ravindran B (2018) Hypergraph clustering: a modularity maximization approach. CoRR abs/1812.10869. arxiv:1812.10869
Maaten Lvd, Hinton G (2008) Visualizing data using t-SNE. J Mach Learn Res 9:2579–2605
Maleki S, Wall D, Pingali K (2021) Netvec: a scalable hypergraph embedding system. CoRR. arxiv:2103.09660
Newman MEJ (2003) The structure and function of complex networks. SIAM Rev 45(2):167–256
Newman MEJ (2006) Modularity and community structure in networks. Proc Natl Acad Sci 103(23):8577–8582. https://doi.org/10.1073/pnas.0601602103
Peng C, Xiao W, Jian P, Z, W (2017) A survey on network embedding. CoRR abs/1711.08752. arxiv:1711.08752
Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '14. Association for Computing Machinery, New York, NY, USA, pp 701–710. https://doi.org/10.1145/2623330.2623732
Takeshi Y, Kazumi S, Naonori U (2003) Cross-entropy directed embedding of network data. In: Proceedings of the 20th international conference on machine learning. AAAI Press, pp 832–839
Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q (2015) Line: large-scale information network embedding. In: Proceedings of the 24th international conference on world wide web. ACM, pp 1067–1077. https://doi.org/10.1145/2736277.2741093
Torgerson W (1952) Multidimensional scaling: I. Theory and method. Psychometrika 17:401–419
von Luxburg U (2007) A tutorial on spectral clustering. Stat Comput 17(4):395–416
Wang D, Cui P, Zhu W (2016) Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, KDD '16. Association for Computing Machinery, New York, NY, USA, pp 1225–1234. https://doi.org/10.1145/2939672.2939753
Yang D, Qu B, Yang J, Cudre-Mauroux P (2019) Revisiting user mobility and social relationships in LBSNs: a hypergraph embedding approach. In: The World Wide Web conference, WWW '19. Association for Computing Machinery, New York, NY, USA, pp 2147–2157. https://doi.org/10.1145/3308558.3313635
Zhou D, Huang J, Schölkopf B (2006) Learning with hypergraphs: clustering, classification, and embedding. In: Proceedings of the 19th international conference on neural information processing systems, NIPS'06. MIT Press, Cambridge, MA, USA, pp 1601–1608

Publisher's Note