Dopater *et al. Applied Network Science* (2024) 9:28
https://doi.org/10.1007/s41109-024-00639-x

Applied Network Science

RESEARCH Open Access

Network embedding based on DepDist contraction

Emanuel Dopater[1†], Eliska Ochodkova[1*†] and Milos Kudelka[1†]

†Emanuel Dopater, Eliska Ochodkova and Milos Kudelka contributed equally to this work.

*Correspondence: eliska.ochodkova@vsb.cz

[1] Department of Computer Science, VSB - Technical University Ostrava, 17. listopadu 15/2172, 70800 Ostrava, Czech Republic

Abstract

Networks provide an understandable and, in the case of small size, visualizable representation of data, which allows us to obtain essential information about the relationships between pairs of nodes, e.g., their distances. In visualization, networks have an alternative two-dimensional vector representation to which various machine-learning methods can be applied. More generally, networks can be transformed into a low-dimensional space using so-called embedding methods, which bridge the gap between network analysis and traditional machine learning by creating numerical representations that capture the essence of the network structure. In this article, we present a new embedding method that uses non-symmetric dependency to find the distance between nodes and applies an iterative procedure to find a satisfactory distribution of nodes in space. For dimension 2 and the visualization of the result, we demonstrate the method's effectiveness on small networks. For higher dimensions and several larger networks, we present the results of two experiments comparing our results with two well-established methods in the research community, namely node2vec and DeepWalk. The first experiment focuses on a qualitative comparison of the methods, while the second focuses on applying and comparing the classification results to embeddings in a higher dimension. Although the presented method does not outperform the two chosen methods, its results are still comparable. Therefore, we also explain the limitations of our method and a possible way to overcome them.

Keywords: Network embedding, Non-symmetric dependency, Dependency distance

Introduction

Finding an efficient representation of network data is crucial for effective network data processing, as traditional representations of network data can be computationally intensive, have low parallelizability, or can be difficult, if not impossible, to use machine learning methods (Cui et al. 2018). Traditional network representations, for example, need to store information about the relationships between pairs of nodes in addition to nodes, which can be a problem, especially in dense, large-scale networks.

Therefore, much attention is now being paid to developing new methods for network embedding, i.e., transforming the original network space into a low-dimensional vector space. Network embedding supports network processing and analysis, such as

community (cluster) detection, network visualization, or link prediction, and it allows us to make general machine-learning techniques applicable to networks.

The basic problem is to find a mapping function between these two spaces. In the network embedding space, the relationships between nodes are captured by the distances between nodes in the vector space, and the structural properties of a node are encoded in its embedding vector. For the embedding space, there are two requirements for network embedding. The first is that the original network can be reconstructed from the learned embedding space. If there is an edge between two nodes, then to preserve the relationships in the network, their distance in the embedding space must be relatively small. Second, the learned embedding space should effectively support network inference, such as predicting unseen links or identifying important nodes (Cui et al. 2018).

In this article, we introduce a novel network embedding method *DepDist Contraction* based on calculating the distance between pairs of network nodes based on their structurally non-symmetric relationship. Our goal is to show that this non-symmetry can be successfully exploited for visualization and network embedding and that symmetric node distances can be found by gradually balancing their non-symmetric relations. After reviewing the methods relevant to our research, we provide background information on a previously published analysis of non-symmetric relationships between pairs of network nodes. Our DepDist Contraction method utilizes this relationship, which is described in the next section. It is an iterative procedure that generates a network embedding and quickly reveals the community structure using the distance defined by the non-symmetric dependency. We then perform experiments on four well-known small networks and show and visually evaluate the results of applying our method to dimension 2. Next, we perform two experiments on several larger networks to compare our method with two well-known network embedding methods. Finally, we discuss the limitations of the DepDist Contraction method and explain possible future improvements. This article is an extended version of a paper presented at the *Complex Networks and Their Applications 2023* (Dopater et al. 2023).

Related work

Different models can be used to transform networks from the original network space to the embedding one, working with different types of information or addressing different goals. Commonly used models include matrix factorization, deep learning with and without random walks, graph kernels, and others.

As a pioneering result, we can mention the locally linear embedding (LLE) method (Roweis and Saul 2000) based on the factorization of the neighborhood matrix, which preserves the similarity between nodes. Method (Ahmed et al. 2013) directly factorizes the proximity matrices with respect to the presence of each edge. Singular Value Decomposition is used due to its optimality for the low-rank approximation of the adjacency matrix (Qiu et al. 2018) and non-negative matrix factorization is often used due to its advantages as an additive model (Wang et al. 2017a).

Existing random walk-based embedding methods attempt to learn node embedding that preserves either structural similarity or node proximity information. By performing a truncated random walk, the network is transformed into sequences of nodes, i.e., paths that preserve the structural proximity of the network. If a node is considered as

a word, the random walk can be considered as a sentence, and the neighborhood of the node can be identified using a measure of co-occurrence. SkipGram (Mikolov et al. 2013) is a famous deep model for neuro-linguistic programming that embeds words in a low-dimensional space by incorporating the context of words in sentences. The SkipGram-inspired method DeepWalk (Perozzi et al. 2014) treats paths as sentences and implements SkipGram to learn the embedding of each node.

With a design principle similar to DeepWalk the node2vec (Perozzi et al. 2014) method is proposed. However, it proposes more flexible random walk strategy, It improves the random walk generation in DeepWalk and mirrors the depth and breadth sampling properties to enhance the network embedding effect.

Finally, let us mention deep neural networks and their variants because they are a suitable choice if we are looking for an efficient model for learning nonlinear functions (Berahmand et al. 2024). Representative methods include SDNE (Wang et al. 2016) or, e.g., SiNE (Wang et al. 2017b). To capture the highly nonlinear structure of the network, SDNE works with a semi-supervised deep model that has multiple layers of nonlinear functions. The model jointly uses first-order and second-order proximity to characterize the local and global structure of the network. Other approaches to network embedding may include (Sulyok and Palla 2023), Salha-Galvan et al. (2022), Park et al. (2020) or e.g. the Yoo et al. (2022) approach, in which the authors deal with embedding methods in directed networks.

One of the important applications for network embedding is the visualization of a network in two-dimensional space. We can find a comparison of visualization results with different embedding approaches in Liao et al. (2018). Classes of graph drawing algorithms, including multi-level and dimensionality reduction-based techniques, are described in detail in a review (Gibson et al. 2013). In network analysis fields, interpretation and understanding of network structure may be based on calculating local or global measures. Visual representation of network structure can help detect, understand, and identify unexpected patterns or outliers in networks.

The layout and arrangement of nodes affect how the user perceives relationships in the network. There is no one best way; the layout of a network depends on which network features are important to us. These may be, for example, specific measures of centrality or important properties of nodes or edges. Criteria for evaluating a network layout include the algorithm's computational complexity, the network's size, the algorithm's ability to follow certain layout rules or aesthetics, clustering, etc.

Many of the methods used are based on the force-directed network paradigm, a paradigm of modeling the network as a physical object system where nodes attract and repel according to some force. Other network drawing algorithms are methods using multi-level and dimensionality reduction-based techniques.

There are two approaches to force-directed layouts: those based on spring embedding and those that solve optimization problems. A very often used method of this type is the method of Fruchterman and Reingold (1991), the connected nodes attract each other while all other nodes, modeled as electrical charges, repel each other.

The second approach considers the layout problem as an optimization problem that minimizes an energy function designed concerning the properties of the network being visualized. Important energy-based techniques are Noack's LinLog (Noack 2007) and

ForceAtlas (Jacomy et al. 2014) layouts. Noack's edge repulsion model removes the bias of the node model towards attraction by ensuring that nodes that are strongly attracting are also strongly repelling, similarly for nodes with weak attraction. Therefore, nodes with a high degree are less likely to be clustered in the center of the network, and it is able to show any underlying clustering structure in the network. ForceAtlas is strongly associated with Noack's LinLog. Its advantage is that all nodes are subject to at least some repulsive force, and poorly connected nodes are thus approximated by well-connected nodes, reducing visual clutter. The forces in the algorithm vary between Noack's edge repulsion model and the Fruchterman and Reingold distributions.

Multilevel algorithms are one of the options that can be used to streamline force-directed techniques. Their idea is to find a sequence of coarser representations of the network, optimize the drawing in the coarsest representation, and propagate this distribution back to the original network. The coarser representations are created by composing connected nodes whose edges become the union of the edges of all the nodes (Hu 2005).

Other options for drawing networks are dimension reduction techniques, including multidimensional scaling, linear dimension reduction (Civril et al. 2006), or spectral graph drawing approaches. The challenge is to preserve the information in a high-dimensional space and capture it in a lower-dimensional representation. Most dimension reduction techniques used for network layout use the graph-theoretical distance between nodes, Freeman (2005), as the information to be preserved.

As mentioned above, the most common use cases of node embedding are visualization, clustering, and link prediction. The problem of visualizing networks in 2D, with its long history, and network drawing algorithms are probably the most well-known embedding techniques commonly used to visualize networks in 2D space. Data-driven network layouts, such as spring embedding, are unsupervised methods of arranging nodes based on their connectivity and are de facto dimensionality reduction techniques. Despite the great potential, layouts are rarely the basis of systematic network visualization.

Therefore, node embedding offers a powerful new paradigm for network visualization: because nodes are mapped to real-valued vectors, researchers can easily leverage general techniques for visualizing high-dimensional data. For example, node embedding can be combined with well-known techniques such as t-SNE (Van der Maaten and Hinton 2008) to create 2D network visualizations (Tang et al. 2015) that can be useful for revealing communities and other hidden structures.

Non-symmetric structural dependency

Structural dependency (hereafter referred to as dependency) is a non-symmetric relationship between pairs of nodes that applies to both weighted and unweighted networks (Kudelka et al. 2019). In our experiments, we work with both types of networks; however, these are always undirected. For this article, we formulate the dependency in a slightly different way. First, let us establish a way to determine the weight $w(A, B, X)$ of the relation between two nodes of the network A, B given their common neighbor X. Let $w(A, X)$ be the weight of the edge between nodes A, X and similarly $w(B, X)$ be the weight of the edge between nodes B, X. Then:

$$w(A,B,X) = w(B,A,X) = \frac{w(A,X) \cdot w(B,X)}{w(A,X) + w(B,X)} \tag{1}$$

The weight defined in this way is half of the harmonic mean, i.e. if the values $w(A, X)$ and $w(B, X)$ are balanced, then the weight $w(A, B, X)$ is around half of $w(A, X)$ and $w(B, X)$ respectively. If, on the other hand, they are not balanced and at least one of the weights $w(A, X)$, $w(B, X)$ is close to zero, then the weight $w(A, B, X)$ is also close to zero.

Now, let us define the strength of the relation between the nodes A, B. If a pair of nodes A, B has multiple common neighbors X_i, then the strength of the relationship between them (the dependency of one on the other) is affected not only by the weights of the edge between these nodes but also by the weights $w(A, B, X_i)$. Therefore, let us define the dependency $D(A, B)$ of a node A on a node B as follows:

$$D(A,B) = \frac{w(A,B) + \sum_{X_i \in \Gamma(A,B)} w(A,B,X_i)}{\sum_{X_j \in N(A)} w(A,X_j)}, \tag{2}$$

where $\Gamma(A, B)$ is the set of common neighbors of nodes A, B and $N(A)$ is the neighborhood (set of all neighbors) of node A.

If there is an edge between nodes A, B, then $w(A, B)$ is the weight of this edge; otherwise, $w(A, B) = 0$. Thus, the dependency is non-zero if and only if the nodes A, B have an edge or at least one common neighbor. A dependency defined in this way is non-symmetric, so $D(A, B) = D(B, A)$ generally does not hold. While the value of the numerator is the same in both directions of the dependency, the value of the denominator may be different. Therefore, it may be true that the dependencies between the nodes of A, B may be substantially different.

Informally speaking, dependency is high if a node is significantly connected to its neighbor through common neighbors compared to the rest of its neighbors. This property provides information about the network's community structure since nodes in a community should have stronger dependencies with each other than with nodes outside the community.

Distance based on non-symmetric dependency

The unanswered question is what distance the two nodes of the network should be if we want to start from the exact dependencies. Two situations can arise: (1) nodes have zero dependencies, and thus neither an edge nor a common neighbor, and (2) nodes have non-zero, potentially non-symmetric dependencies. In the first case, we have no direct information to determine the distance. In the second case, we have to convert the dependencies into Euclidean space, which is, by definition, symmetric. For further considerations, let us start with a simple interpretation of the dependency, which can be described as a relation that attracts two nodes together. To express this relation, we define the mutual dependency coefficient $q_S(A, B)$ as the product of the partial dependencies of the nodes A, B with their arithmetic mean, i.e:

$$q_S(A,B) = D(A,B) \cdot D(B,A) \cdot \frac{D(A,B) + D(B,A)}{2} \tag{3}$$

The coefficient q takes into account both dependencies and, thanks to the average, information about their balance. The coefficient q_s can also be further used to determine the

symmetric distance between nodes A, B. An alternative is to work with the non-symmetric distance and leave the determination of the symmetric distance to the iterative procedure described in Sect. 4. In this case, the assumed distance between the nodes may be non-symmetric at the input. For this case, let us define the coefficient $q_N(A,B)$:

$$q_N(A,B) = D(A,B)^2 \cdot D(B,A) \tag{4}$$

The values of $q_S(A,B), q_N(A,B)$ are from the interval [0, 1] and severely penalize situations where at least one of the dependencies is very low. Our experiments show that using both alternatives of the q coefficient provides the same result. However, the symmetric version is more effective in revealing communities, and on the other hand, the non-symmetric version stabilizes the resulting embedding better. Therefore, for the basic version of the algorithm, $q(A,B) = q_N(A,B)$ will hold for the following.

Now we can define the maximum distance *maxDepDist* between pairs of network nodes, from which we can derive the distance of node A from node B with non-zero dependencies as follows:

$$DepDist(A,B) = (1 - q(A,B)) \cdot maxDepDist \tag{5}$$

Note that we cannot determine this distance based on non-symmetric dependency (*DepDist* for short) between absolutely independent nodes. In the following, we will show that we can still use such an incompletely formulated distance for network embedding.

DepDist contraction

As mentioned above, the essence of network embedding is to find a network representation in low-dimensional space in which the relationships between network nodes are highly preserved. We next present an iterative procedure based on a straightforward use of DepDist that provides such a representation. We refer to this procedure as DepDist Contraction,[1] and this is because its essence is to bring pairs of nodes closer together so that the result is close to the distance based on their mutual dependencies.

Even though we are concerned with network embedding and the presented procedure is independent of the chosen dimension, we focus our experiments only on dimension 2. This allows us to visualize the result of the DepDist Contraction and, at least, assess it visually.

Remark In the following, we will use the term node A to mean both a node of the network and a point representing that node in n-dimensional space.

Algorithm

The first step of the algorithm to find the representation of the network in n-dimensional space is to randomly distribute the points representing each network node into a cube of dimension n with edge length a. Next, we set the value of *maxDepDist* to be much smaller than the edge length a (so there is enough space for contraction). We then iterate so that in one iteration, each node A moves in space to some node B (we will return to

[1] Non-parallel Python implementation used for the experiments with small networks is at https://github.com/emaneldopater/DepDistContraction/tree/conference.

the selection of node B later). For the move, the step length corresponds to the distance between A and B and their coefficient $q(A, B)$. The iteration ends, as we will show later, either after a fixed number of steps or after the contraction has stabilized.

One step of iteration

Let us consider a node A, a node B selected for it, their coefficient $q(A, B)$, and their expected $DepDist(A, B)$. By one iteration step, we mean moving node A to node B so that their distance approaches $DepDist(A, B)$. Let $\hat{u}$ be a unit vector in the direction of the vector $B - A$. The new position A' of node A is:

$$A' = A + acc(A, B) \cdot (\parallel B - A \parallel - DepDist(A, B)) \cdot \hat{u} \tag{6}$$

The function $acc(q(A, B))$ changes the effect of the coefficient $q(A, B)$ on the length of the move. The function is designed to increase the move length significantly when the distance between nodes A, B is too large (above some defined threshold), i.e., when $\parallel B - A \parallel$ is significantly greater than $DepDist(A, B)$. On the other hand, the move length decreases with decreasing distance of nodes, which gradually stabilizes node positions in space when node positions change negligibly. Therefore, we define a maximum threshold distance for acceleration $maxAccDist > maxDepDist$. Next, for each pair of nodes A, B, we determine the threshold distance for acceleration:

$$accDist(A, B) = (1 - q(A, B)) \cdot maxAccDist \tag{7}$$

Based on this threshold distance for acceleration, we then define the acceleration coefficient $accCoef$ to be equal to one for $accDist(A, B) = \parallel B - A \parallel$:

$$accCoef(A, B) = 0.5 + 0.5 \cdot \frac{\|B-A\|}{accDist(A,B)} \tag{8}$$

The acc function is then defined as:

$$acc(A, B) = q(A, B)^{\frac{1}{accCoef(A,B)}} \tag{9}$$

Thus, in general, pairs of nodes that are weakly dependent on each other are farther apart than strongly dependent nodes, slowly converging to the expected distances $DepDist(A, B)$ and $DepDist(B, A)$, respectively. For example, two high-degree nodes that share a common edge but have very few common neighbors compared to their other neighbors will hardly change their position during an iteration. In contrast, for example, nodes that are part of a large and almost disjoint clique have strong dependencies and, thus, small distances to neighbors will move very quickly.

More interesting is the situation where node A is strongly dependent on node B, but the reverse is not true, e.g., node B is a hub, and node A has degree 1. Using the non-symmetric alternative $q_N(A, B)$, node A will approach node B very quickly, and node B will move slowly towards node A.

Selecting node to move

In an iteration, for each node A, a different node B is selected, to which node A is moved according to the procedure described above; it is, therefore, necessary to determine how node B can be selected. As described above, non-zero dependency can only be

computed for nodes with a common edge or at least one common neighbor; therefore, this assumption limits the selection. For DepDist Contraction, we use a strategy based on the assumption that the fewer neighbors a node has, the less information we have about its neighborhood, and we should "look further." For node A, we therefore set the probability of randomly selecting its neighbor B to be related to the degree of node A:

$$p(A) = 1 - \frac{1}{1+k(A)}, \tag{10}$$

where $k(A)$ is the degree of node A; with complementary probability $1 - p(A)$, a neighbor of the neighbors of A is then chosen at random. Thus, if a node has a very high degree, its neighbor is chosen with near certainty, and conversely, if a node has degree $k(A) = 1$, then its neighbor is chosen with probability 0.5.

When to stop iterating

The algorithm depends on only three parameters: (1) the edge length a of the n-dimensional cube into which the nodes of the network are randomly distributed at the beginning, (2) the maximum expected distance *maxDepDist* of the nodes in the embedding from which the distances between each pair of nodes are derived, and (3) the maximum distance *maxAccDist* for the acceleration of the move from which the distance above which the move accelerates and below which it decreases is derived for each pair of nodes. Thus, from the perspective of the whole network, it is a contraction that leads to a distribution of nodes in a small part of the input n-dimensional cube where the nodes almost stop moving. Our experiments show that, regardless of the size of the network, after 20–50 iterations, the community structure emerges (strongly dependent groups of nodes), and after 200–500 iterations, the distribution changes very little; groups of strongly dependent nodes move (relatively) away from each other, and the distribution stabilizes. Thus, the number of iterations needed is not much affected by the network size because the algorithm efficiently separates locally strongly connected substructures from the rest of the network. The strength of the DepDist Contraction algorithm is, therefore, most evident when applied to networks with significant community structure, and its discovery in embedding is only a side effect of the dependency distance.

Figure 1 visualizes the distribution of nodes after 50 iterations of the four networks we used in our experiments; it is a 2D embedding, which is complemented for better clarity by the edges between the nodes and the sizes of the nodes corresponding to their degree. As can be seen, even after a relatively small number of iterations, the community structure of the networks is obvious.

Scalability

Computing the dependency of one node on another is similar to computing the clustering coefficient and has time complexity $O(k^2)$, where k is the average degree of the network (we can compute the dependencies in both directions simultaneously). However, the computations for each pair of nodes are independent and can be computed in parallel as needed. Within a single iteration, it is possible to store the current node positions at the beginning and compute the dependencies, including moving the nodes to their new positions in parallel. At the end of the iteration, the current positions are swapped

Table 1 Properties of the four small networks

Network	N	M	k	min_k	max_k	CC	Q
Karate	34	78	4.588	1	17	0.588	0.415
Lesmis	77	254	6.597	1	36	0.736	0.551
Footbal	115	613	10.661	7	12	0.403	0.604
Netscience	379	914	4.823	1	34	0.798	0.845

with the new ones. Thus, during the algorithm, we work with two states of the network (current and new node positions); therefore, the spatial complexity is $O(N)$, where N is the number of nodes in the network.

To estimate the time complexity, we assume a sparse network where the relationship between the number of edges and the number of nodes is $O(N)$. If we want to optimize the computational complexity, we need to compute the dependencies continuously during the iterative procedure (i.e., only when necessary) and store them for reuse. Thus, the estimate of the time complexity of computing all dependencies is based on the dependency computation complexity and the total number of node pairs for which the dependency needs to be computed. Given that we compute dependencies for neighbors and neighbors of neighbors based on random selection, we can estimate the time complexity of computing all required dependencies to be $O(Nk^4)$; however, this is the worst case, assuming that dependencies are computed for all neighbors and neighbors of neighbors for all nodes in the network. Furthermore, for sparse networks, the spatial complexity changes to $O(Nk^2)$ in the case of stored dependencies. Again, this is the worst case, which does not occur in practice since dependencies with neighbors of neighbors are rarely computed for higher degree nodes (see "Selecting node to move" section). In general, for sparse networks, we can expect a time and space complexity of $O(Nk^3)$ and $O(Nk)$, respectively.

The random selection around the selected node depends on the representation of the network. If we use an adjacency list, then neighbor selection has complexity $O(1)$. Therefore, for the total complexity, we only need to consider the number of iterations r; the estimate of the total time complexity is then $O(rN + Nk^3)$ for sparse networks.

Example: 2D embedding and visualization

In order to demonstrate the effectiveness of the DepDist Contraction algorithm, we used four small networks; for these small networks, the quality of the embedding can be visually assessed in the form of a visualized network layout. We chose well-known networks from Mark E.J. Newman[2]: Zachary's karate club (karate), Les Miserables (lesmis), American College football (football), giant component of Coauthorships in network science (netscience). In Table 1, we can see that each network has different properties (number of nodes and edges, average, minimum and maximum degree, average clustering coefficient, Louvain modularity (Blondel et al. 2008).

In this example, we used dimension $n = 2$ for all four networks, the side size of the square for the random initial nodes distribution $a = 1$, i.e., a square with a diagonal

[2] http://www-personal.umich.edu/~mejn/netdata/.

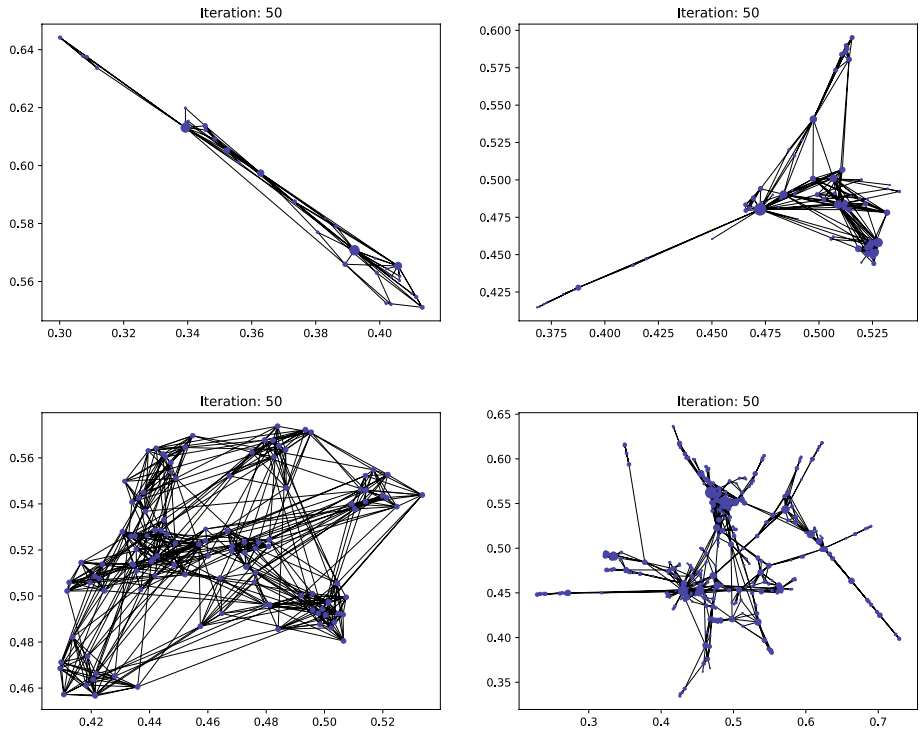

Fig. 1 2D embedding for karate, lesmis, football, netscience (giant component) networks after 50 iterations

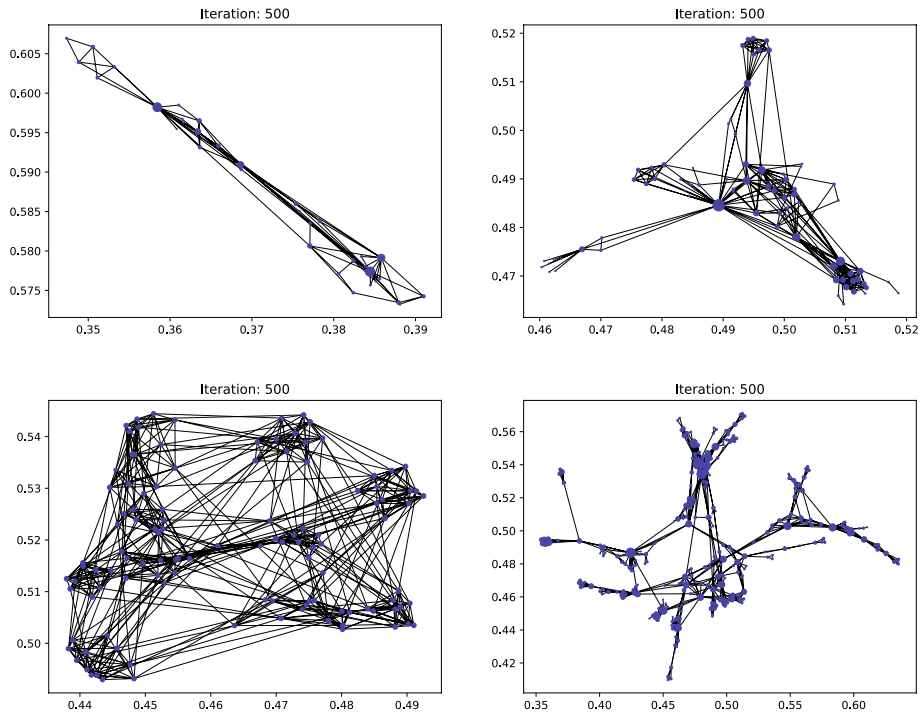

Fig. 2 2D embedding for karate, lesmis, football, netscience (giant component) networks after 500 iterations

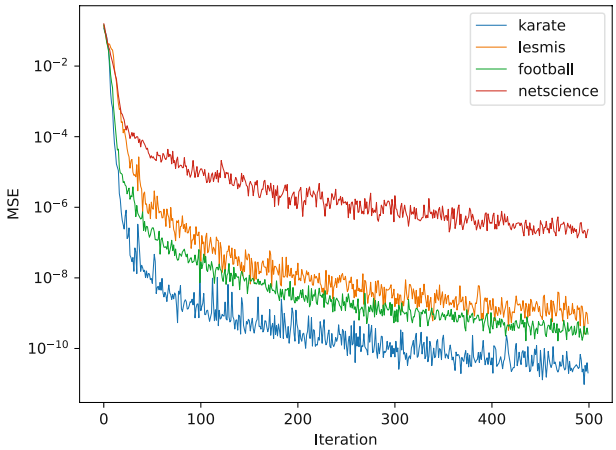

Fig. 3 Evolution of MSE between two consecutive iterations for karate, lesmis, footbal, netscience (giant component) networks.is significantly larger

[0, 0], [1, 1], the maximum expected dependency distance $maxDepDist = 0.002$, and the maximum acceleration distance $maxAccDist = 0.01$. The result of applying the DepDist Contraction algorithm is shown for 50 and 500 iterations in Figs. 1 and 2; the difference between 200 and 500 iterations is visually negligible, and there is virtually no further movement. Figure 3 shows the changes in the positions of the network nodes expressed in terms of the mean squared error (MSE) between two consecutive iterations.

Although our goal is embedding (i.e., in this case, transforming the network into a vector representation of dimension 2), the result is comparable to layout-oriented algorithms, which typically use balancing based on attractive and repulsive forces between pairs of nodes. However, compared to the force-directed layout in Fig. 4, there is a significant difference in the karate layout. Namely, the dependency is much more related to the connectivity of the nodes to the neighborhood than to the edge weights. Therefore, the distance between pairs of nodes is only small when both dependencies are high. On the other hand, if at least one dependency goes to zero, the distance increases, regardless of the edge weights. In Fig. 2, this property in the karate network highlights (1) the separation of the three groups of nodes in the center and at the boundaries and (2) the relatively large distances between nodes that are weakly connected to their neighborhood.

The figures show how embedding is affected by other characteristics of each network. Zachary's karate club contains no major cliques except triangles; Les Miserables contains well-separated cliques, near-cliques, and star-like substructures; American college football contains several near-clique structures and no hubs; Coauthorships in network science contain many small, clearly separated cliques clustered around hubs of various sizes.

Observations and heuristics

While observing the progress of the embedding generation algorithm in small networks, we noticed four points that sometimes had a negative effect on the result. Four heuristics that resulted from these observations are described below:

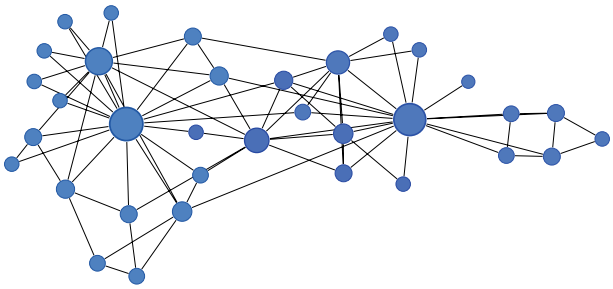

Fig. 4 Force-directed layout of karate club network

1. Creation of a *super-node* that connects all nodes to avoid the problem of small components.
2. Swap q-functions to quickly reveal communities.
3. Initialization of higher degree nodes near the sides of n-dimensional cubes.
4. Estimation of the parameters *maxDepDist* and *maxAccDist*.

Heuristics 1: super-node

In our experiments, we encountered a problem where the network was disconnected, and small isolated components remained in the same place. For this case, we artificially added an extra super-node to the network when initializing the algorithm and connected every node in the network to this node with a small edge weight; this procedure ensured that the network was always connected. The super-node is placed exactly in the center of the n-dimensional cube.

Heuristics 2: q-function swap

Observing the algorithm's progress, it can be seen that the symmetric version of the function q_S generally causes larger moves than the non-symmetric version q_N. Therefore, it is better to start with the symmetric version and use the non-symmetric version after the communities have separated, which causes only small moves of hubs and tunes the distances within the communities. In the experiments below, we swapped after one hundred iterations out of a total of five hundred.

Heuristics 3: random initialization

The basic version of the algorithm assumes a uniform random distribution of nodes at initialization. This is sufficient in most cases, but there may be situations where high-degree nodes, i.e., hubs, are initialized close together at the beginning. This can negatively affect the resulting embedding if the nodes remain very close during iterations. The heuristic more likely distributes hubs near the side of the n-dimensional cube. As a result, nodes with low degrees tend to be closer to the cube's center and hubs vice versa. This solution significantly reduces this negative effect.

Heuristics 4: maxDepDist and maxAccDist estimation

The estimation of the parameter *maxDepDist* is based on the expected size of the N dimensional space in which the points corresponding to the nodes of the network will be located after the embedding process. We use a cube of dimension n with side length one for the initial random placement of nodes. For simplicity, let's assume that the points are uniformly distributed within this cube. The distance between adjacent pairs of points in this cube will then be approximately $\frac{1}{N^{1/n}}$. Due to contraction, this distance must be much smaller in the resulting embedding. Experiments have shown that a good estimate for *maxDepDist* is one-hundredth of this distance. The acceleration distance must then be proportionally larger; for all settings, we set $maxAccDist = 10 \cdot maxDepDist$ in experiments.

Experiments

The comparison of network embedding methods is a generally formulated task related to the purpose for which the embedding is needed. In our case, the goal of embedding is to project the original network into a non-structural representation in a low-dimensional space while preserving structural properties. One possible solution is to apply machine learning methods for selected tasks (e.g. clustering and classification) to the resulting embeddings and compare their results with our knowledge about the network (e.g. communities or node roles); these methods can also be combined. Comparing the quality of embeddings is a challenging and ambiguous task that is still under research.

In our experiments, we applied two approaches; the first uses a framework combining multi-factor evaluation described below, and the second is based on the classification of structurally detected roles. The experiments and their evaluation were performed in Python using the libraries *sklearn*[3] a *pytorch*.[4] We compared parallel *C++* implementation of the DepDist Contraction method with two well-known methods that are based on slightly different approaches:

DeepWalk This embedding method is based on random walks. The first step is to generate random walks, which can be thought of as sequences. The second step is the generation of the embedding; it uses the same logic as *word2vec* with sequences or words. Initially, the embedding of nodes is random, and during training, the embedding is adjusted by back-propagation. The embedding of nodes that are close together in a sequence is adjusted so that they are close together in the resulting embedding space.

node2vec Its logic is very similar to DeepWalk, but it uses 2 additional parameters, p and q. The parameter p (return parameter) controls the probability that the node will immediately visit the node from which it came. The parameter q (in-out parameter) controls whether the selection of nodes should be more breadth-first or depth-first. By setting these two parameters, it is possible to generate embeddings either for communities (nodes in the same communities will tend to have more similar embeddings) or for structural roles (nodes with similar roles in the network, e.g. hubs, will

[3] https://scikit-learn.org/stable/.

[4] https://pytorch.org/.

Table 2 Properties of 8 networks used in experiments

Network	Mouse	Airport	Email	Ca Hep	Ca Grqc	FB	Lastfm	P2P
Nodes	1029	464	986	9875	5241	4039	7624	6301
Edges	1700	7595	16017	25996	14495	88234	27806	20777
Density	0.003	0.070	0.032	0.001	0.001	0.010	0.001	0.001
Max degree	153	175	342	65	81	1045	216	97
Min degree	1	1	1	1	1	1	1	1
Avg degree	3.304	32.737	32.489	5.265	5.531	43.691	7.294	6.595
Assortat.	−0.215	−0.055	−0.025	0.267	0.659	0.063	0.017	0.035
Clust. coeff	0	0.476	0.266	0.472	0.530	0.606	0.220	0.011
Triangles	0	100358	104395	28339	48260	1612010	40433	2383
Max k-core	5	50	34	31	43	115	20	10
Components	20	2	1	427	354	1	1	2

tend to have similar embeddings). node2vec is generally considered to be the best embedding algorithm, but a drawback may be finding optimal parameter settings.

CGE framework

As mentioned above, evaluating the embedding quality is not an easy task, and the results may not be as expected; there are not many tools available. We have chosen *CGE framework* in the version recommended by its authors (Dehghan-Kooshkghazi et al. 2022). This framework is suitable for unsupervised comparison of different network embedding methods and can compare network embeddings in any dimension.

The input to the evaluation algorithm is the network connectivity (in the form of an edge list or adjacency list) and the output is a "divergence" score. This divergence score indicates the embedding quality; a lower value indicates higher quality, but has no quantitative meaning. Thus, comparing embeddings only provides a ranking of the embeddings.

The *CGE* framework includes two main steps for embedding generation. First, it uses a graph clustering algorithm [specifically the ensemble clustering algorithm for graphs (Poulin and Théberge 2019)] based on the Louvain algorithm and consensus clustering. This first step identifies dense parts of the graph, creating stable clusters, meaning that more edges are captured inside than outside the communities. These clusters provide a good macroscopic view of the network. The second step computes the expected number of edges within each cluster found in the first step and also between them. The whole embedding is scored by calculating a divergence score between this expected number of edges and the actual number of edges present in the network.

Experimental networks

We performed the following two experiments with a total of eight networks of different types. Details are below, and their properties are listed in Table 2.

Mouse Brain (Mouse) network represents a brain connectivity graph derived from the somatosensory cortex of a mouse. The nodes are

	individual neurons, and the edges represent synapses. Available at https://networkrepository.com/bn-mouse-kasthuri-graph-v4.php
Airports	networks represent information about flights between airports based on a record of more than 3.5 million domestic US flights from 1990 to 2009. Nodes represent airports and edges connections between airports. We used it from *CGE* (Dehghan-Kooshkghazi et al. 2022) framework article. Available at https://github.com/ftheberge/GraphMiningNotebooks/tree/master/Datasets/Airports
Emails (Email)	networks represent email data from a large European research institution. Nodes are users, and edges represent the sent emails. Available at https://snap.stanford.edu/data/email-Eu-core-temporal.html
Ca-HepTh (Ca Hep)	network is a collaboration network from arXiv's High Energy Physics—Theory section. Nodes represent authors and edges are coauthorships between authors. Available at https://networkrepository.com/ca-HepTh.php
Ca-GrQq (Ca Grqq)	network is similar to *Ca-HepTh* network, but it captures the research section of general relativity and quantum cosmology. Available at https://networkrepository.com/ca-GrQc.php
Facebook combined (FB)	network represents circles of users (nodes) and their friends (edges). Available at https://snap.stanford.edu/data/ego-Facebook.html
LastFM Asia (Lastfm)	network is part of a dataset from the music streaming service LastFM, particularly focusing on user interactions within Asia. Nodes represent users, and edges indicate mutual follower relationships. Available at https://snap.stanford.edu/data/index.html#socnets
Gnutella P2P (P2P)	network maps the topology of connections in the Gnutella peer-to-peer file-sharing network as of August 2008. Nodes represent hosts in the network and edges are the connections between them. Available at: https://snap.stanford.edu/data/p2p-Gnutella08.html

Comparison using CGE framework

The comparison was performed for all eight networks, three algorithms (DepDist Contraction, neode2vec, DeepWalk), and a total of six embedding dimensions (4, 6, 8, 16, 32, 64, 128). For each network-method-dimension combination, we generated embeddings in thirty independent runs; the settings for DeepWalk and node2vec were taken from the CGE article (Dehghan-Kooshkghazi et al. 2022). We evaluated each embedding using

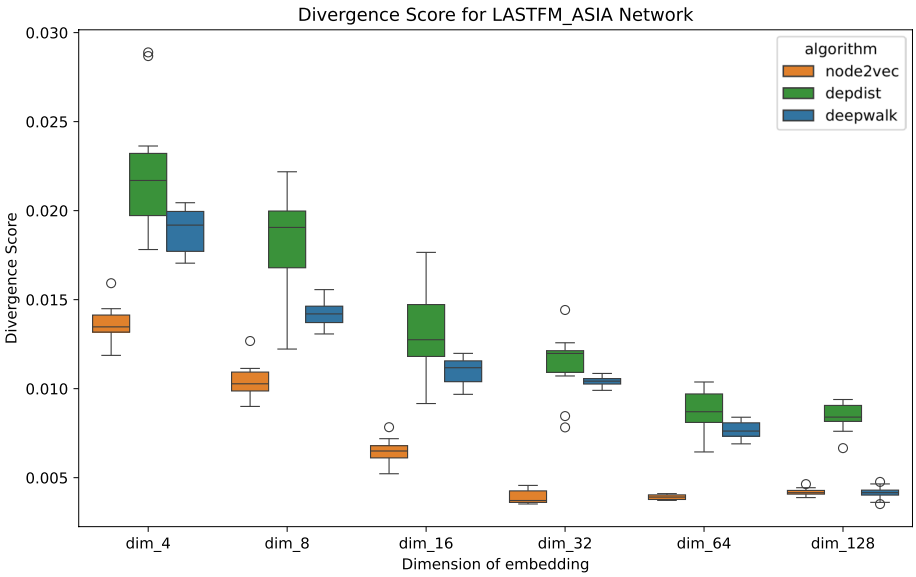

Fig. 5 Divergence scores for Lastfm_ASIA network

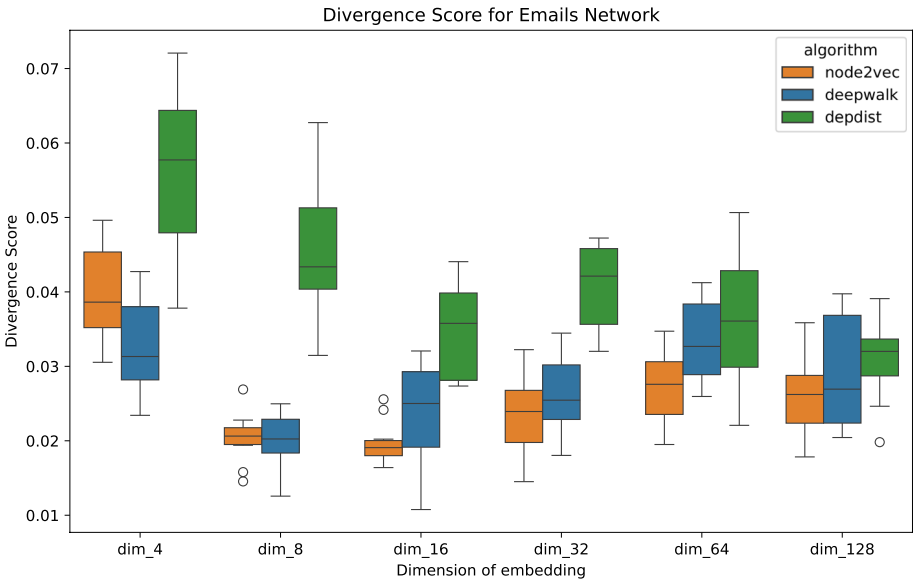

Fig. 6 Divergence scores for Emails network

the *CGE* framework[5] and obtained results showing the divergence score of the methods, including their stability, in Figs. 5, 6, 7 and 8 with boxplots.

In all eight network-method combinations, node2vec was predictably the most successful. However, the four figures show different cases. For Lastfm, the results for all three methods are comparable. For the Email network, these methods are the least stable, with DeepWalk and node2vec having the best results for dimensions 8 and 16,

[5] https://github.com/KrainskiL/CGE.jl.

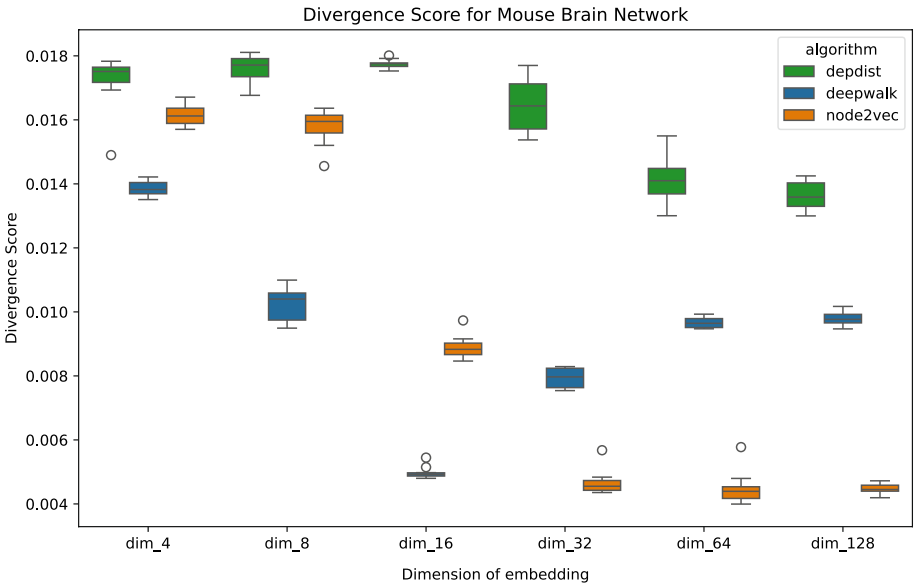

Fig. 7 Divergence scores for Mouse Brain network

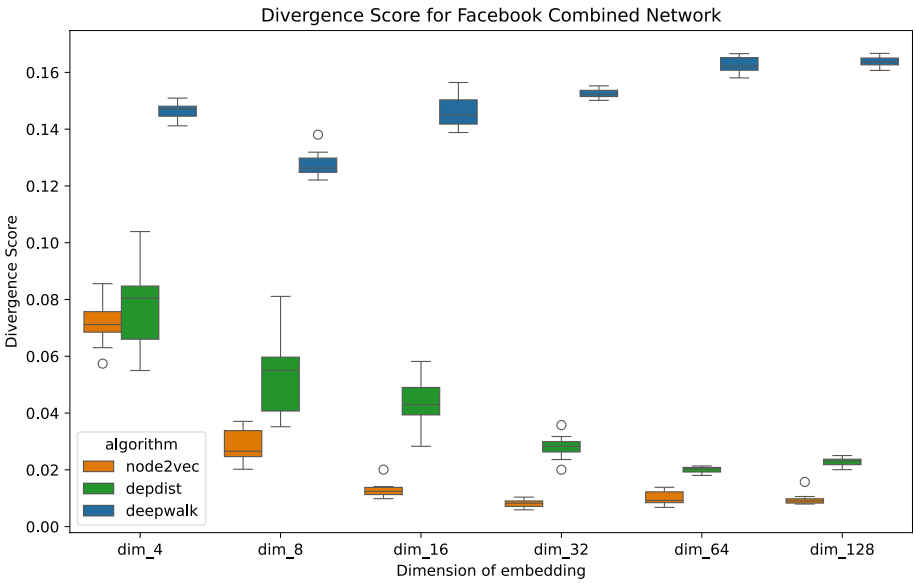

Fig. 8 Divergence scores for Facebook combined network

while for DepDist Contraction, it is dimension 128. In the remaining two cases, the results are the most stable, with DepDisp Contraction for FB network being comparable to node2vec, unlike DeepWalk. For Mouse, our method is the worst, but DeepWalk does not have good results either.

The quality of embedding is obviously affected by the nature of the network. For example, some networks tend to have multiple connected components or well-separated communities, while other types of networks do not; it may also depend on other

Table 3 Comparison of rankings (medians) for the eight networks studied

Method	dim	Mouse	Airports	Email	Ca Hep	Ca Grqc	FB	Lastfm	P2P
DeepWalk	4	0.014	0.003	0.031	0.041	0.035	0.147	0.019	0.006
	8	0.010	0.002	0.020	0.038	0.029	0.126	0.014	0.006
	16	0.005	0.002	0.025	0.029	0.014	0.145	0.011	0.004
	32	0.008	0.003	0.025	0.013	0.012	0.152	0.010	0.004
	64	0.010	0.003	0.033	0.018	0.024	0.162	0.008	0.005
	128	0.010	0.003	0.027	0.025	0.031	0.164	0.004	0.006
DepDist	4	0.018	0.006	0.058	0.044	0.038	0.080	0.022	0.006
	8	0.018	0.007	0.043	0.045	0.037	0.055	0.019	0.006
	16	0.018	0.006	0.036	0.045	0.037	0.043	0.013	0.006
	32	0.016	0.006	0.042	0.045	0.038	0.028	0.012	0.006
	64	0.014	0.007	0.036	0.045	0.037	0.020	0.009	0.006
	128	0.014	0.007	0.032	0.044	0.038	0.023	0.008	0.006
node2vec	4	0.016	0.002	0.039	0.043	0.034	0.071	0.013	0.006
	8	0.016	0.002	0.021	0.039	0.029	0.027	0.010	0.006
	16	0.009	0.002	0.019	0.035	0.015	0.012	0.006	0.006
	32	0.005	0.003	0.024	0.017	0.006	0.008	0.004	0.003
	64	0.004	0.004	0.028	0.009	0.005	0.009	0.004	0.003
	128	0.004	0.005	0.026	0.008	0.006	0.009	0.004	0.003

properties, such as density or assortativity. All of these factors can affect embedding algorithms in different ways.

Table 3 shows the medians of rankings for all eight networks under examination. It can be observed that DeepWalk and our method were not successful in the Ca Hep and Ca grqc networks. Table 2 indicates that the cause of this outcome is likely due to the high assortativity and the large number of isolated connected components present in these networks. DeepWalk was unsuccessful in FB network, which is related to the high average degree and the specific connectivity with a large number of triangles. In contrast, the low success rate of our method in the Mouse network is attributable to the absence of triangles in this network.

Classification of roles

Several aspects were considered when comparing the quality of embedding in the previous section. In this section, we focus on a typical machine learning task, classification. Note that the result of applying any of the network embedding methods is a vector dataset to which classification can be applied in a straightforward manner. However, to do this, we need to partition the individual vectors representing the network nodes into classes and assign a label to each network node. For this task, we used the approach of Kudelka et al. (2019), which defines three types of structurally detected roles based on the analysis of the dependencies around a given node.

First, we need to binarize the dependency relationship so that node (A) *is dependent on* node (B) if ($D(A,B) \geq 0.5$). Then the roles are defined as follows:

A strongly prominent node	is not dependent on any of its neighbors, and at least one of its neighbors is dependent on it.
A weakly prominent node	has at least one neighbor that is dependent on it, and the node itself is not dependent on that neighbor.
A non-prominent node	is a node that is neither strongly nor weakly prominent.

Note that roles defined in this way require a fairly detailed analysis of the node surroundings to capture the information needed for their detection. Some of this

Table 4 Frequency of roles in the 8 networks studied

Network	Sum	Strongly prominent	Strongly prominent (%)	Weakly prominent	Weakly prominent (%)	Non prominent	Non prominent (%)
Mouse	1029	134	13.0	0	0.0	895	87.0
Airports	464	123	26.5	40	8.6	301	64.9
Email	986	308	31.2	15	1.5	663	67.2
Ca Hep	9875	2421	24.5	741	7.5	6713	68.0
Ca Grqc	5241	1023	19.5	445	8.5	3773	72.0
FB	4039	61	1.5	1242	30.8	2736	67.7
Lastfm	7624	2246	29.5	366	4.8	5012	65.7
P2P	6301	1577	25.0	1	0.0	4723	75.0

information is necessarily lost during embedding generation. It is, therefore, useful to compare how the three methods deal with this loss. Furthermore, the occurrence of individual roles is highly unbalanced, and the relative frequency of their occurrence also varies considerably depending on the type of network.

In Dopater and Kudělka (2022), we applied graph neural networks for the role classification task, which also worked with structural information. The roles served as labels for the nodes and were used to fit the classifier. Classification accuracy was very high in this case.

We first determined structural roles for all eight networks to assess whether and how the embeddings generated by each method could be used for classification. The frequency of occurrence of roles in each of the networks studied is highly unbalanced as can be seen in Table 4. Strongly prominent nodes are hubs that usually connect communities, weakly prominent nodes are important nodes in communities, and non-prominent nodes are all other nodes in the networks.

We then worked only with 128-dimensional embeddings, so our classifier was a simple linear neural network with an input layer of 128 dimensions, a next layer of 256 and 512 dimensions, and an output layer of 3 (because of the 3 output classes). The activation function was ReLU, except for the softmax output layer. There was a 20% dropout layer between each layer to avoid overfitting. We used a learning rate of 0.001 and multiple training epochs of 25. We then performed an independent classification with 5-fold cross-validation for each combination of *network-method*. We used the F1 score with macro-averaging (because of the multiple classes) as a metric.

The results are shown in Fig. 9 and vary due to an unbalanced number of node role classes. The best classification is for the Mouse and P2P networks (although stability is lower). Weakly prominent nodes are (almost) absent in both networks, and apparently, the structural difference between roles that needs to be detected in classification disappears after embedding. However, the results were expected to be quite disappointing; the differences between the methods are insignificant, and neither captures the structural roles of the nodes very well. The cause is, therefore, general and related to the fact that the distances and locally dense groups of nodes in the embedding space do not provide enough information to analyze local structural relationships in depth.

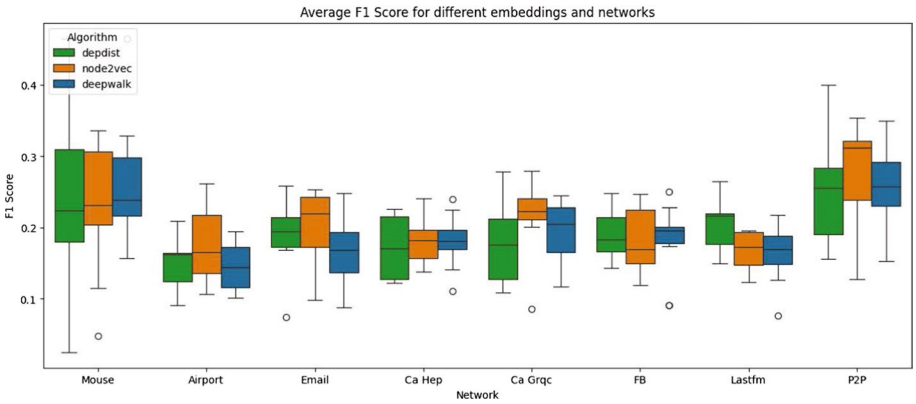

Fig. 9 Results of role classification in the 8 networks studied

Limitations

The main reason why the DepDist Contraction method gives slightly worse results than the two methods compared is the calculation of the distances between the network nodes. Unlike the relatively long random walks in both algorithms, our method only computes distances between neighboring nodes and nodes within distance 2 due to the use of non-symmetric dependency. Interestingly (and this can be seen even when visualizing small networks), it still reveals the community structure and hubs very well and quickly. However, the problem grows as the density increases and the network's community structure weakens. This problem can be addressed by modifying the dependency computation even for distances greater than two. Such a modification is relatively straightforward but beyond the scope of this article.

The problem associated with dense networks is the complexity of the dependency computation, which limits the application of our method in the setting of dense large-scale networks with very high-degree nodes. A modification of the computation can be based on dependency estimation, where it is not necessary to examine the entire neighborhood of the pair of nodes (e.g., sampling and random walks can be used); this modification is also beyond the scope of this article. It is necessary to add that in the case of our experiments with C++ parallel implementation, the embedding computation time is maximum in seconds.

We have not yet been able to estimate the number of iterations needed to stabilize the embedding. We assumed that it is possible to define the relationship between a sufficiently stable embedding, its parameters (dimension n, cube edge length a, $maxDepDist$), and the network size. Experiments, while not confirming this, have shown that for small networks, lower hundreds of iterations are sufficient for embedding stabilization (see Fig. 3). For all experiments, we set the number of iterations to 500.

Conclusion

This article introduces a novel method of network embedding. The DepDist Contraction algorithm presented in this work is based on a simple iterative procedure and utilizes a previously published non-symmetric dependency between pairs of nodes. This method's application provides embedding results that are comparable to two

well-known methods in most experiments performed, including experiments with small, straightforwardly visualizable networks. It should be noted that the method has certain limitations, which are primarily due to the computation of distances between nodes in the resulting embedding space. This is based on the analysis of only small distances in the network. Another related technical limitation is the complexity of computing the non-symmetric dependency. Consequently, the presented method is currently unsuitable for networks with high-density and frequent nodes with degrees close to the network's number of nodes. Nevertheless, we posit that the DepDist Contraction method has considerable potential and that the two mentioned limitations can be addressed without compromising the fundamental nature of the algorithm. Both of these will be the subject of future research.

Author contributions
The authors contributed equally to the work.

Funding
This work is supported by SGS, VSB-Technical University of Ostrava, under Grant No. SP2024/007.

Data availability
There are no restrictions to the availability of data.

Declarations

Competing interests
The authors declare no competing interests.

Received: 3 May 2024 Accepted: 25 June 2024
Published online: 02 July 2024

References

Ahmed A, Shervashidze N, Narayanamurthy S, Josifovski V, Smola AJ (2013) Distributed large-scale natural graph factorization. In: Proceedings of the 22nd international conference on world wide web, pp 37–48
Berahmand K, Daneshfar F, Salehi ES, Li Y, Xu Y (2024) Autoencoders and their applications in machine learning: a survey. Artif Intell Rev 57(2):28
Blondel VD, Guillaume J-L, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. J Stat Mech: Theory Exp 2008(10):10008
Civril A, Magdon-Ismail M, Bocek-Rivele E (2006) SSDE: fast graph drawing using sampled spectral distance embedding. In: International symposium on graph drawing. Springer, Berlin, pp 30–41
Cui P, Wang X, Pei J, Zhu W (2018) A survey on network embedding. IEEE Trans Knowl Data Eng 31(5):833–852
Dehghan-Kooshkghazi A, Kaminski B, Krainski Pralat P, Théberge F (2022) Evaluating node embeddings of complex networks
Dopater E, Kudělka M (2022) Node classification based on non-symmetric dependencies and graph neural networks. In: International conference on complex networks and their applications. Springer, Berlin, pp 347–357
Dopater E, Ochodkova E, Kudelka M (2023) Network embedding based on depdist contraction. In: International conference on complex networks and their applications. Springer, Berlin, pp 427–439
Freeman LC (2005) Graphic techniques for exploring social network data. Models Methods Soc Netw Anal 28:248–269
Fruchterman TM, Reingold EM (1991) Graph drawing by force-directed placement. Softw: Pract Exp 21(11):1129–1164
Gibson H, Faith J, Vickers P (2013) A survey of two-dimensional graph layout techniques for information visualisation. Inf Vis 12(3–4):324–357
Hu Y (2005) Efficient, high-quality force-directed graph drawing. Math J 10(1):37–71
Jacomy M, Venturini T, Heymann S, Bastian M (2014) Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software. PLoS ONE 9(6):98679
Kudelka M, Ochodkova E, Zehnalova S, Plesnik J (2019) Ego-zones: non-symmetric dependencies reveal network groups with large and dense overlaps. Appl Netw Sci 4(1):1–49
Liao L, He X, Zhang H, Chua T-S (2018) Attributed social network embedding. IEEE Trans Knowl Data Eng 30(12):2257–2270
Maaten L, Hinton G (2008) Visualizing data using T-SNE. J Mach Learn Res 9(11):2579–2605
Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems, vol 26

Noack A (2007) Energy models for graph clustering. J. Graph Algorithms Appl. 11(2):453–480
Park C, Kim D, Han J, Yu H (2020) Unsupervised attributed multiplex network embedding. In: Proceedings of the AAAI conference on artificial intelligence, vol 34, pp 5371–5378
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, pp 701–710
Poulin V, Théberge F (2019) Ensemble clustering for graphs: comparisons and applications. Appl Netw Sci 4(1):1–13
Qiu J, Dong Y, Ma H, Li J, Wang K, Tang J (2018) Network embedding as matrix factorization: unifying deepwalk, line, pte, and node2vec. In: Proceedings of the eleventh ACM international conference on web search and data mining, pp 459–467
Roweis ST, Saul LK (2000) Nonlinear dimensionality reduction by locally linear embedding. science 290(5500):2323–2326
Salha-Galvan G, Lutzeyer JF, Dasoulas G, Hennequin R, Vazirgiannis M (2022) Modularity-aware graph autoencoders for joint community detection and link prediction. Neural Netw 153:474–495
Sulyok B, Palla G (2023) Greedy routing optimisation in hyperbolic networks. Sci Rep 13(1):23026
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, pp 1067–1077
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, pp 1225–1234
Wang X, Cui P, Wang J, Pei J, Zhu W, Yang S (2017a) Community preserving network embedding. In: Proceedings of the AAAI conference on artificial intelligence, vol 31
Wang S, Tang J, Aggarwal C, Chang Y, Liu H (2017b) Signed network embedding in social media. In: Proceedings of the 2017 SIAM international conference on data mining. SIAM, pp 327–335
Yoo H, Lee Y-C, Shin K, Kim S-W (2022) Directed network embedding with virtual negative edges. In: Proceedings of the fifteenth ACM international conference on web search and data mining, pp 1291–1299

Publisher's Note