4
Graph Embedding

4.1 Introduction

Graph embedding aims to map each node in a given graph into a low-dimensional vector representation (or commonly known as node embedding) that typically preserves some key information of the node in the original graph. A node in a graph can be viewed from two domains: 1) the original graph domain, where nodes are connected via edges (or the graph structure); and 2) the embedding domain, where each node is represented as a continuous vector. Thus, from this two-domain perspective, graph embedding targets on mapping each node from the graph domain to the embedding domain so that the information in the graph domain can be preserved in the embedding domain. Two key questions naturally arise: 1) what information to preserve? and 2) how to preserve this information? Different graph embedding algorithms often provide different answers to these two questions. For the first question, many types of information have been investigated such as node's neighborhood information (Perozzi et al., 2014; Tang et al., 2015; Grover and Leskovec, 2016), node's structural role (Ribeiro et al., 2017), node status (Ma et al., 2017; Lai et al., 2017; Gu et al., 2018) and community information (Wang et al., 2017c). There are various methods proposed to answer the second question. While the technical details of these methods vary, most of them share the same idea, which is to reconstruct the graph domain information to be preserved by using the node representations in the embedding domain. The intuition is those good node representations should be able to reconstruct the information we desire to preserve. Therefore, the mapping can be learned by minimizing the reconstruction error. We illustrate an overall framework in Figure 4.1 to summarize the general process of graph embedding. As shown in Figure 4.1, there are four key components in the general framework as:

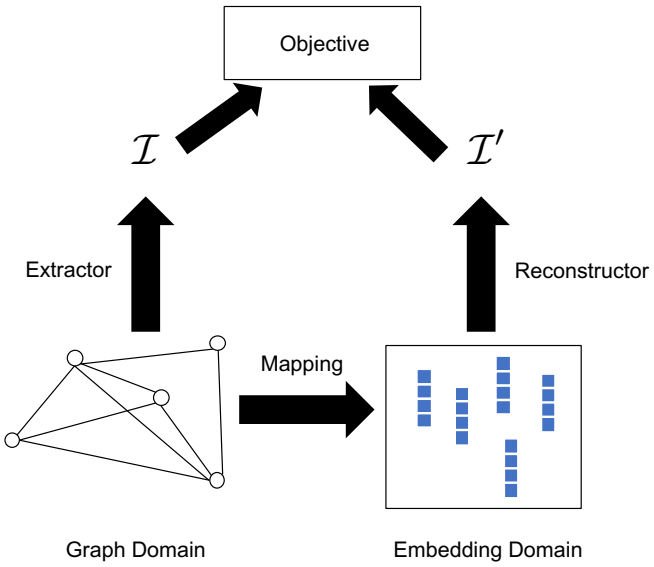


Figure 4.1 A general framework for graph embedding

- A mapping function, which maps the node from the graph domain to the embedding domain.
- An information extractor, which extracts the key information $\mathcal{I}$ we want to preserve from the graph domain.
- A reconstructor to construct the extracted graph information $\mathcal{I}$ using the embeddings from the embedding domain. Note that the reconstructed information is denoted as $\mathcal{I}'$ as shown in Figure 4.1.
- An objective based on the extracted information $\mathcal{I}$ and the reconstructed information $\mathcal{I}'$. Typically, we optimize the objective to learn all parameters involved in the mapping and/or reconstructor.

In this chapter, we introduce representative graph embedding methods, which preserve different types of information in the graph domain, based on the general framework in Figure 4.1. Furthermore, we introduce graph embedding algorithms designed specifically for complex graphs, including heterogeneous graphs, bipartite graphs, multi-dimensional graphs, signed graphs, hypergraphs, and dynamic graphs.

4.2 Graph Embedding on Simple Graphs

In this section, we introduce graph embedding algorithms for simple graphs that are static, undirected, unsigned, and homogeneous, as introduced in Chapter 2.2. We organize algorithms according to the information they attempt to preserve, including node co-occurrence, structural role, node status, and community structure.

4.2.1 Preserving Node Co-occurrence

One of the most popular ways to extract node co-occurrence in a graph is via performing random walks. Nodes are considered similar to each other if they tend to co-occur in these random walks. The mapping function is optimized so that the learned node representations can reconstruct the "similarity" extracted from random walks. One representative network embedding algorithm preserving node co-occurrence is DeepWalk (Perozzi et al., 2014). Next, we first introduce the DeepWalk algorithm under the general framework by detailing its mapping function, extractor, reconstructor, and objective. Then, we present more node co-occurrence preserving algorithms such as node2vec (Grover and Leskovec, 2016) and LINE (Tang et al., 2015).

Mapping Function

A direct way to define the mapping function $f(v_i)$ is using a look-up table. It means that we retrieve node v_i's embedding $\mathbf{u}_i$ given its index i. Specifically, the mapping function is implemented as:

$$f(v_i) = \mathbf{u}_i = \mathbf{e}_i^\top \mathbf{W}, \tag{4.1}$$

where $\mathbf{e}_i \in \{0, 1\}^N$ with $N = |\mathcal{V}|$ is the one-hot encoding of the node v_i. In particular, $\mathbf{e}_i$ contains a single element $\mathbf{e}_i[i] = 1$ and all other elements are 0. $\mathbf{W}^{N \times d}$ is the embedding parameters to be learned where d is the dimension of the embedding. The i-th row of the matrix $\mathbf{W}$ denotes the representation (or the embedding) of node v_i. Hence, the number of parameters in the mapping function is $N \times d$.

Random Walk Based Co-occurrence Extractor

Given a starting node $v^{(0)}$ in a graph $\mathcal{G}$, we randomly walk to one of its neighbors. We repeat this process from the node until T nodes are visited. This random sequence of visited nodes is a random walk of length T on the graph. We formally define a random walk as follows.

Definition 4.1 (Random Walk) Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ denote a connected graph. We now consider a random walk starting at node $v^{(0)} \in \mathcal{V}$ on the graph G. Assume that at the t-th step of the random walk, we are at node $v^{(t)}$ and then we proceed the random walk by choosing the next node according to the following probability:

$$p(v^{(t+1)}|v^{(t)}) = \begin{cases} \frac{1}{d(v^{(t)})}, & \text{if } v^{(t+1)} \in \mathcal{N}(v^{(t)}) \\ 0, & \text{otherwise,} \end{cases}$$

where $d(v^{(t)})$ denotes the degree of node $v^{(t)}$ and $\mathcal{N}(v^{(t)})$ is the set of neighbors of $v^{(t)}$. In other words, the next node is randomly selected from the neighbors of the current node following a uniform distribution.

We use a random walk generator to summarize the above process as below:

$$\mathcal{W} = \text{RW}(\mathcal{G}, v^{(0)}, T),$$

where $\mathcal{W} = (v^{(0)}, \ldots, v^{(T-1)})$ denotes the generated random walk where $v^{(0)}$ is the starting node and T is the length of the random walk.

Random walks have been employed as a similarity measure in various tasks such as content recommendation (Fouss et al., 2007) and community detection (Andersen et al., 2006). In DeepWalk, a set of short random walks is generated from a given graph, and then node co-occurrence is extracted from these random walks. Next, we detail the process of generating the set of random walks and extracting co-occurrence from them.

To generate random walks that can capture the information of the entire graph, each node is considered as a starting node to generate γ random walks. Therefore, there are $N \cdot \gamma$ random walks in total. This process is shown in Algorithm 1. The input of the algorithm includes a graph $\mathcal{G}$, the length T of the random walk, and the number of random walks γ for each starting node. From line 4 to line 8 in Algorithm 1, we generate γ random walks for each node in $\mathcal{V}$ and add these random walks to $\mathcal{R}$. In the end, $\mathcal{R}$, which consists of $N \cdot \gamma$ generated random walks, is the output of the algorithm.

Algorithm 1: Generating Random Walks

```
1 Input: 𝒢 = {𝒱, ℰ}, T, γ
2 Output: ℛ
3 Initialization: ℛ ← ∅
4 for i in range(1,γ) do
5 |   for v ∈ 𝒱 do
6 |   |   𝒲 ← RW(𝒢, v^(0), T)
7 |   |   ℛ ← ℛ ∪ {𝒲}
8 |   end
9 end
```

These random walks can be treated as sentences in an "artificial language" where the set of nodes $\mathcal{V}$ is its vocabulary. The Skip-gram algorithm (Mikolov et al., 2013) in language modeling tries to preserve the information of the sentences by capturing the co-occurrence relations between words in these sentences. For a given center word in a sentence, those words within a certain distance w away from the center word are treated as its "context". Then the center word is considered to be co-occurred with all words in its "context". The Skip-gram algorithm aims to preserve such co-occurrence information. These concepts are adopted to the random walks to extract co-occurrence relations between nodes (Perozzi et al., 2014). Specifically, we denote the co-occurrence of two nodes as a tuple (v_{con}, v_{cen}), where v_{cen} denotes the center node and v_{con} indicates one of its context nodes. The process of extracting the co-occurrence relations between nodes from the random walks is shown in Algorithm 2. For each random walk $\mathcal{W} \in \mathcal{R}$, we iterate over the nodes in the random walk (line 5). For each node $v^{(i)}$, we add $(v^{(i-j)}, v^{(i)})$ and $(v^{(i+j)}, v^{(i)})$ into the list of co-occurrence $\mathcal{I}$ for $j = 1, \ldots, w$ (from line 6 to line 9). Note that for the cases where $i - j$ or $i + j$ is out of the range of the random walk, we simply ignore them. For a given center node, we treat all its "context" nodes equally regardless of the distance between them. In (Cao et al., 2015), the "context" nodes are treated differently according to their distance to the center node.

Algorithm 2: Extracting Co-occurrence

Input: $\mathcal{R}$, w
Output: $\mathcal{I}$
Initialization: $\mathcal{I} \leftarrow []$
for $\mathcal{W}$ *in* $\mathcal{R}$ **do**
  for $v^{(i)} \in \mathcal{W}$ **do**
    for *j in range(1,w)* **do**
      $\mathcal{I}.append((v^{(i-j)}, v^{(i)}))$
      $\mathcal{I}.append((v^{(i+j)}, v^{(i)}))$
    end
  end
end

Reconstructor and Objective

With the mapping function and the node co-occurrence information, we discuss the process of reconstructing the co-occurrence information using the representations in the embedding domain. To reconstruct the co-occurrence information, we try to infer the probability of observing the tuples in $\mathcal{I}$. For any given tuple $(v_{con}, v_{cen}) \in \mathcal{I}$, there are two roles of nodes, i.e., the center node v_{cen} and the context node v_{con}. A node can play both roles, i.e., the center node and the context node of other nodes. Hence, two mapping functions are employed to generate two node representations for each node corresponding to its two roles. They can be formally stated as:

$$f_{cen}(v_i) = \mathbf{u}_i = e_i^\top \mathbf{W}_{cen}$$
$$f_{con}(v_i) = \mathbf{v}_i = e_i^\top \mathbf{W}_{con}.$$

For a tuple (v_{con}, v_{cen}), the co-occurrence relation can be explained as observing v_{con} in the context of the center node v_{cen}. With the two mapping functions f_{cen} and f_{con}, the probability of observing v_{con} in the context of v_{cen} can be modeled using a softmax function as follows:

$$p(v_{con}|v_{cen}) = \frac{exp(f_{con}(v_{con})^\top f_{cen}(v_{cen}))}{\sum_{v \in \mathcal{V}} exp(f_{con}(v) f_{cen}^\top(v_{cen}))}, \tag{4.2}$$

which can be regarded as the reconstructed information from the embedding domain for the tuple (v_{con}, v_{cen}). For any given tuple (v_{con}, v_{cen}), the reconstructor *Rec* can return the probability in Eq. (4.2) that is summarized as:

$$p(v_{con}|v_{cen}) = Rec((v_{con}, v_{cen})).$$

If we can accurately infer the original graph information of $\mathcal{I}$ from the embedding domain, the extracted information $\mathcal{I}$ can be considered as well-reconstructed. To achieve the goal, the *Rec* function should return high probabilities for extracted tuples in the $\mathcal{I}$, while low probabilities for randomly generated tuples. We assume that these tuples in the co-occurrence $\mathcal{I}$ are independent to each other as that in the Skip-gram algorithm (Mikolov et al., 2013). Hence, the probability of reconstructing $\mathcal{I}$ can be modeled as follows:

$$\mathcal{I}' = Rec(\mathcal{I}) = \prod_{(v_{con}, v_{cen}) \in \mathcal{I}} p(v_{con}|v_{cen}), \tag{4.3}$$

There may exist duplicate tuples in $\mathcal{I}$. To remove these duplicates in Eq. (4.3), we re-formulate it as follows:

$$\prod_{(v_{con}, v_{cen}) \in set(\mathcal{I})} p(v_{con}|v_{cen})^{\#(v_{con}, v_{cen})}, \tag{4.4}$$

where $set(\mathcal{I})$ denotes the set of unique tuples in $\mathcal{I}$ without duplicates and $\#(v_{con}, v_{cen})$ is the frequency of tuples (v_{con}, v_{cen}) in $\mathcal{I}$. Therefore, the tuples that are more frequent in $\mathcal{I}$ contribute more to the overall probability in Eq. (4.4). To ensure better reconstruction, we need to learn the parameters of the mapping functions such that Eq. (4.4) can be maximized. Thus, the node embeddings $\mathbf{W}_{con}$ and $\mathbf{W}_{cen}$ (or parameters of the two mapping functions) can be learned by minimizing the following objective:

$$\mathcal{L}(\mathbf{W}_{con}, \mathbf{W}_{cen}) = -\sum_{(v_{con}, v_{cen}) \in set(\mathcal{I})} \#(v_{con}, v_{cen}) \cdot \log p(v_{con}|v_{cen}), \tag{4.5}$$

where the objective is the negative logarithm of Eq. (4.4).

Speeding Up the Learning Process

In practice, calculating the probability in Eq. (4.2) is computationally unfeasible due to the summation over all nodes in the denominator. To address this challenge, two main techniques have been employed – one is hierarchical softmax, and the other is negative sampling (Mikolov et al., 2013).

Hierarchical Softmax

In the hierarchical softmax, nodes in a graph $\mathcal{G}$ are assigned to the leaves of a binary tree. A toy example of the binary tree for hierarchical softmax is shown in Figure 4.2 where there are 8 leaf nodes, i.e., there are 8 nodes in the original graph $\mathcal{G}$. The probability $p(v_{con}|v_{cen})$ can now be modeled through the path to node v_{con} in the binary tree. For example, the path to node v_3 is highlighted in red. Given the path to the node v_{con} identified by a sequence of

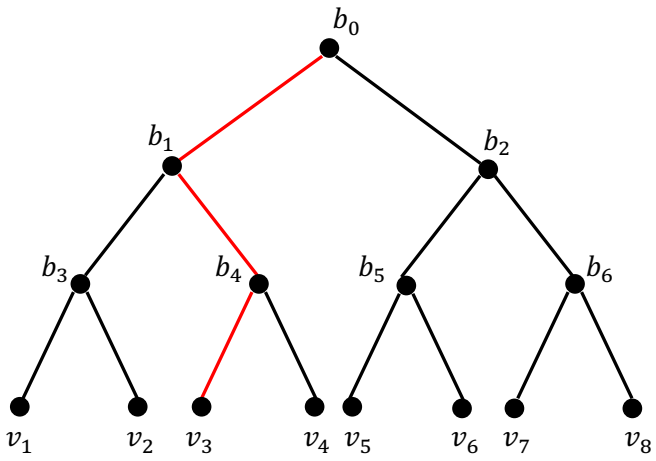


Figure 4.2 An illustrative example of hierarchical softmax. The path to node v_3 is highlighted in red.

tree nodes $(p^{(0)}, p^{(1)}, \ldots, p^{(H)})$ with $p^{(0)} = b_0$ (the root) and $p^{(H)} = v_{con}$, the probability can be obtained as:

$$p(v_{con}|v_{cen}) = \prod_{h=1}^{H} p_{path}(p^{(h)}|v_{cen}),$$

where $p_{path}(p^{(h)}|v_{cen})$ can be modeled as a binary classifier that takes the center node representation $f(v_{cen})$ as input. Specifically, for each internal node, a binary classifier is built to determine the next node for the path to proceed.

We use the root node b_0 to illustrate the binary classifier where we are calculating the probability $p(v_3|v_8)$ (i.e. $(v_{con}, v_{cen}) = (v_3, v_8)$)) for the toy example shown in Figure 4.2. At the root node b_0, the probability of proceeding to the left node can be computed as:

$$p(left|b_0, v_8) = \sigma(f_b(b_0)^\top f(v_8)),$$

where f_b is a mapping function for the internal nodes, f is the mapping function for the leaf nodes (or nodes in graph $\mathcal{G}$) and σ is the sigmoid function. Then the probability of the right node at b_0 can be calculated as

$$p(right|b_0, v_8) = 1 - p(left|b_0, v_8) = \sigma(-f_b(b_0)^\top f(v_8)).$$

Hence, we have

$$p_{path}(b_1|v_8) = p(left|b_0, v_8).$$

Note that the embeddings of the internal nodes can be regarded as the parameters of the binary classifiers, and the input of these binary classifiers is the embedding of the center node in (v_{con}, v_{cen}). By using the hierarchical softmax instead of the conventional softmax in Eq. (4.2), the computational cost can be hugely reduced from $O(|\mathcal{V}|)$ to $O(\log|\mathcal{V}|)$. Note that in the hierarchical softmax, we do not learn two mapping functions for nodes in $\mathcal{V}$ any more. Instead, we learn a mapping function f for nodes in $\mathcal{V}$ (or leaf nodes in the binary tree) and a mapping function f_b for the internal nodes in the binary tree.

Example 4.2 (Hierarchical Softmax) Assume that (v_3, v_8) is a tuple describing co-occurrence information between nodes v_3 and v_8 with v_3 the context node and v_8 the center node in a given graph $\mathcal{G}$ and the binary tree of hierarchical softmax for this graph is shown in Figure 4.2. The probability of observing v_3 in the context of v_8, i.e., $p(v_3|v_8)$, can be computed as follows:

$$\begin{aligned} p(v_3|v_8) &= p_{path}(b_1|v_8) \cdot p_{path}(b_4|v_8) \cdot p_{path}(v_3|v_8) \\ &= p(left|b_0, v_8) \cdot p(right|b_1, v_8) \cdot p(left|b_4, v_8). \end{aligned}$$

Negative Sampling

Another popular approach to speed up the learning process is negative sampling (Mikolov et al., 2013). It is simplified from Noise Contrasitive Estimation (NCE) (Gutmann and Hyvärinen, 2012) that has been shown to approximately maximize the log probability of the softmax function. However, our ultimate goal is to learn high quality node representations instead of maximizing the probabilities. It is reasonable to simplify NCE as long as the learned node representations retain good quality. Hence, the following modifications are made to NCE and Negative Sampling are defined as follows. For each tuple (v_{con}, v_{cen}) in $\mathcal{I}$, we sample k nodes that do not appear in the "context" of the center node v_{cen} to form the negative sample tuples. With these negative sample tuples, we define Negative Sampling for (v_{con}, v_{cen}) by the following objective:

$$\log\sigma\left(f_{con}(v_{con})^\top f_{cen}(v_{cen})\right) + \sum_{i=1}^{k} E_{v_n \sim P_n(v)}\left[\log\sigma\left(-f_{con}(v_n)^\top f_{cen}(v_{cen})\right)\right], \tag{4.6}$$

where the probability distribution $P_n(v)$ is the noise distribution to sample the negative tuples that is often set to $P_n(v) \sim d(v)^{3/4}$ as suggested in (Mikolov et al., 2013; Tang et al., 2015). By maximizing Eq. (4.6), the probabilities between the nodes in the true tuples from $\mathcal{I}$ are maximized while these between the sample nodes in the negative tuples are minimized. Thus, it tends to ensure that the learned node representations preserve the co-occurrence information.

The objective in Eq. (4.6) is used to replace $\log p(v_{con}|v_{cen})$ in Eq. (4.5) that results in the following overall objective:

$$\mathcal{L}(\mathbf{W}_{con}, \mathbf{W}_{cen}) = \sum_{(v_{con}, v_{cen}) \in set(\mathcal{I})} \#(v_{con}, v_{cen}) \cdot (\log \sigma\left(f_{con}(v_{con})^\top f_{cen}(v_{cen})\right) + \sum_{i=1}^{k} E_{v_n \sim P_n(v)} \left[\log \sigma\left(-f_{con}(v_n)^\top f_{cen}(v_{cen})\right)\right]). \tag{4.7}$$

By using negative sampling instead of the conventional softmax, the computational cost can be hugely reduced from $O(|\mathcal{V}|)$ to $O(k)$.

Training Process in Practice

We have introduced the overall objective function in Eq. (4.5) and two strategies to improve the efficiency of calculating the loss function. The node representations can now be learned by optimizing the objective in Eq. (4.5) (or its alternatives). However, in practice, instead of evaluating the entire objective function over the whole set of $\mathcal{I}$ and performing gradient descent based updates, the learning process is usually done in a batch-wise way. Specifically, after generating each random walk $\mathcal{W}$, we can extract its corresponding co-occurrence information $\mathcal{I}_{\mathcal{W}}$. Then, we can formulate an objective function based on $\mathcal{I}_{\mathcal{W}}$ and evaluate the gradient based on this objective function to perform the updates for the involved node representations.

Other Co-occurrence Preserving Methods

There are some other methods that aim to preserve co-occurrence information such as node2vec (Grover and Leskovec, 2016) and LINE (second-order) (Tang et al., 2015). They are slightly different from DeepWalk but can still be fitted to the general framework in Figure 4.1. Next, we introduce these methods with the focus on their differences from DeepWalk.

node2vec

node2vec (Grover and Leskovec, 2016) introduces a more flexible way to explore the neighborhood of a given node through the biased-random walk, which is used to replace the random walk in DeepWalk to generate $\mathcal{I}$. Specifically, a second-order random walk with two parameters p and q is proposed. It is defined as follows:

Definition 4.3 Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ denote a connected graph. We consider a random walk starting at node $v^{(0)} \in \mathcal{V}$ in the graph G. Assume that the random walk has just walked from the node $v^{(t-1)}$ to node $v^{(t)}$ and now resides at the

node $v^{(t)}$. The walk needs to decide which node to go for the next step. Instead of choosing $v^{(t+1)}$ uniformly from the neighbors of $v^{(t)}$, a probability to sample is defined based on both $v^{(t)}$ and $v^{(t-1)}$. In particular, an unnormalized "probability" to choose the next node is defined as follows:

$$\alpha_{pq}(v^{(t+1)}|v^{(t-1)}, v^{(t)}) = \begin{cases} \frac{1}{p} & \text{if } \mathrm{dis}(v^{(t-1)}, v^{(t+1)}) = 0 \\ 1 & \text{if } \mathrm{dis}(v^{(t-1)}, v^{(t+1)}) = 1 \\ \frac{1}{q} & \text{if } \mathrm{dis}(v^{(t-1)}, v^{(t+1)}) = 2 \end{cases} \tag{4.8}$$

where $\mathrm{dis}(v^{(t-1)}, v^{(t+1)})$ measures the length of the shortest path between node $v^{(t-1)}$ and $v^{(t+1)}$. The unnormalized "probability" in Eq. (4.8) can then be normalized as a probability to sample the next node $v^{(t+1)}$.

Note that the random walk based on this normalized probability is called second-order random walk as it considers both the previous node $v^{(t-1)}$ and the current node $v^{(t)}$ when deciding the next node $v^{(t+1)}$. The parameter p controls the probability to revisit the node $v^{(t-1)}$ immediately after stepping to node $v^{(t)}$ from node $v^{(t-1)}$. Specifically, a smaller p encourages the random walk to revisit while a larger p ensures the walk to less likely backtrack to visited nodes. The parameter q allows the walk to differentiate the "inward" and "outward" nodes. When $q < 1$, the walk is biased to nodes that are close to node $v^{(t-1)}$, and when $q > 1$, the walk tends to visit nodes that are distant from node $v^{(t-1)}$. Therefore, by controlling the parameters p and q, we can generate random walks with different focuses. After generating the random walks according to the normalized version of the probability in Eq. (4.8), the remaining steps of node2vec are the same as DeepWalk.

LINE

The objective of LINE (Tang et al., 2015) with the second order proximity can be expressed as follows:

$$\begin{aligned} -\sum_{(v_{con}, v_{cen}) \in \mathcal{E}} & \Big(\log \sigma \left(f_{con}(v_{con})^\top f_{cen}(v_{cen}) \right) \\ & + \sum_{i=1}^{k} E_{v_n \sim P_n(v)} \left[\log \sigma \left(-f_{con}(v_n)^\top f_{cen}(v_{cen}) \right) \right] \Big), \end{aligned} \tag{4.9}$$

where $\mathcal{E}$ is the set of edges in the graph $\mathcal{G}$. Comparing Eq. (4.9) with Eq. (4.7), we can find that the major difference is that LINE adopts $\mathcal{E}$ instead of $\mathcal{I}$ as the information to be reconstructed. In fact, $\mathcal{E}$ can be viewed as a special case of $\mathcal{I}$ where the length of the random walk is set to 1.

A Matrix Factorization View

In (Qiu et al., 2018b), it is shown that these aforementioned network embedding methods can be viewed from a matrix factorization perspective. For example, we have the following theorem for DeepWalk.

Theorem 4.4 ((Qiu et al., 2018b)) *In the matrix form, DeepWalk with negative sampling for a given graph $\mathcal{G}$ is equivalent to factoring the following matrix:*

$$\log\left(\frac{\text{vol}(\mathcal{G})}{T}\left(\sum_{r=1}^{T}\mathbf{P}^r\right)\mathbf{D}^{-1}\right) - \log(k),$$

where $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ with $\mathbf{A}$ the adjacency matrix of graph $\mathcal{G}$ and $\mathbf{D}$ its corresponding degree matrix, T is the length of random walk, $\text{vol}(\mathcal{G}) = \sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}|}\mathbf{A}_{i,j}$ *and k is the number of negative samples.*

Actually, the matrix form of DeepWalk can be also fitted into the general framework introduced in above. Specifically, the information extractor is

$$\log\left(\frac{\text{vol}(\mathcal{G})}{T}\left(\sum_{r=1}^{T}\mathbf{P}^r\right)\mathbf{D}^{-1}\right).$$

The mapping function is the same as that introduced for DeepWalk, where we have two mapping functions, $f_{cen}()$ and $f_{con}()$. The parameters for these two mapping functions are $\mathbf{W}_{cen}$ and $\mathbf{W}_{con}$, which are also the two sets of node representations for the graph $\mathcal{G}$. The reconstructor, in this case, can be represented in the following form: $\mathbf{W}_{con}\mathbf{W}_{cen}^{\top}$. The objective function can then be represented as follows:

$$\mathcal{L}(\mathbf{W}_{con}, \mathbf{W}_{cen}) = \left\|\log\left(\frac{\text{vol}(\mathcal{G})}{T}\left(\sum_{r=1}^{\top}\mathbf{P}^r\right)\mathbf{D}^{-1}\right) - \log(b) - \mathbf{W}_{con}\mathbf{W}_{cen}^{\top}\right\|_F^2.$$

The embeddings $\mathbf{W}_{con}$ and $\mathbf{W}_{cen}$ can thus be learned by minimizing this objective. Similarly, LINE and node2vec can also be represented in the matrix form (Qiu et al., 2018b).

4.2.2 Preserving Structural Role

Two nodes close to each other in the graph domain (e.g., nodes d and e in Figure 4.3) tend to co-occur in many random walks. Therefore, the co-occurrence preserving methods are likely to learn similar representations for these nodes in the embedding domain. However, in many real-world applications, we want to embed the nodes u and v in Figure 4.3 to be close in the embedding domain

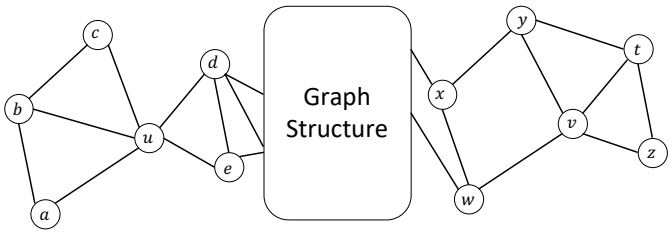


Figure 4.3 An illustration of two nodes that share similar structural role

since they share a similar structural role. For example, if we want to differentiate hubs from non-hubs in airport networks, we need to project the hub cities, which are likely to be apart from each other but share a similar structural role, into similar representations. Therefore, it is vital to develop graph embedding methods that can preserve structural roles.

The method struc2vec is proposed to learn node representations that can preserve structural identity (Ribeiro et al., 2017). It has the same mapping function as DeepWalk while it extracts structural role similarity from the original graph domain. In particular, a degree-based method is proposed to measure the pairwise structural role similarity, which is then adopted to build a new graph. Therefore, the edge in the new graph denotes structural role similarity. Next, the random walk based algorithm is utilized to extract co-occurrence relations from the new graph. Since struc2vec shares the same mapping and reconstructor functions as Deepwalk, we only detail the extractor of struc2vec. It includes the structural similarity measure, the built new graph, and the biased random walk to extract the co-occurrence relations based on the new graph.

Measuring Structural Role Similarity

Intuitively, the degree of nodes can indicate their structural role similarity. In other words, two nodes with similar degree can be considered as structurally similar. Furthermore, if their neighbors also have similar degree, these nodes can be even more similar. Based on this intuition, a hierarchical structural similarity measure is proposed in (Ribeiro et al., 2017). We use $R_k(v)$ to denote the set of nodes that are k-hop away from the node v. We order the nodes in $R_k(v)$ according to their degree to the degree sequence $s(R_k(v))$. Then, the structural distance $g_k(v_1, v_2)$ between two nodes v_1 and v_2 considering their k-hop neighborhoods can be recursively defined as follows:

$$g_k(v_1, v_2) = g_{k-1}(v_1, v_2) + \text{dis}(s(R_k(v_1)), s(R_k(v_2))),$$

where $\text{dis}(s(R_k(v_1)), s(R_k(v_2))) \geq 0$ measures the distance between the ordered degree sequences of v_1 and v_2. In other words, it indicates the degree similarity of k-hop neighbors of v_1 and v_2. Note that $g_{-1}(\cdot,\cdot)$ is initialized with 0. Both $\text{dis}(\cdot,\cdot)$ and $g_k(\cdot,\cdot)$ are distance measures. Therefore, the larger they are, more dissimilar the two compared inputs are. The sequences $s(R_k(v_1))$ and $s(R_k(v_2))$ can be of different lengths and their elements are arbitrary integers. Thus, Dynamic Time Warping (DTW) (Sailer, 1978; Salvador and Chan, 2007) is adopted as the distance function $\text{dis}(\cdot,\cdot)$ since it can deal with sequences with different sizes. The DTW algorithm finds the optimal alignment between two sequences such that the sum of the distance between the aligned elements is minimized. The distance between two elements a and b is measured as:

$$\text{dis}(a,b) = \frac{\max(a,b)}{\min(a,b)} - 1.$$

Note that this distance depends on the ratio between the maximum and minimum of the two elements; thus, it can regard $(1,2)$ much different from $(100,101)$.

Constructing a Graph Based on Structural Similarity

After obtaining the pairwise structural distance, we can construct a multilayer weighted graph that encodes the structural similarity between the nodes. Specifically, with k^* as the diameter of the original graph $\mathcal{G}$, we can build a k^* layer graph where the k-th layer is built upon the weights defined as follows:

$$w_k(u,v) = \exp(-g_k(u,v)).$$

Here, $w_k(u,v)$ denotes the weight of the edge between nodes u and v in the k-th layer of the graph. The connection between nodes u and v is stronger when the distance $g_k(u,v)$ is smaller. Next, we connect different layers in the graph with directed edges. In particular, every node v in the layer k is connected to its corresponding node in the layers $k-1$ and $k+1$. We denote the node v in the k-th layer as $v^{(k)}$ and the edge weights between layers are defined as follows

$$w(v^{(k)}, v^{(k+1)}) = \log(\Gamma_k(v) + e), k = 0, \ldots, k^* - 1,$$
$$w(v^{(k)}, v^{(k-1)}) = 1, k = 1, \ldots, k^*,$$

where

$$\Gamma_k(v) = \sum_{v_j \in \mathcal{V}} \mathbb{1}(w_k(v, v_j) > \bar{w}_k)$$

with $\bar{w}_k = \sum_{(u,v)\in\mathcal{E}_k} w_k(u,v) / \binom{N}{2}$ denoting the average edge weight of the complete graph ($\mathcal{E}_k$ is its set of edges) in the layer k. Thus, $\Gamma_k(v)$ measures the similarity of node v to other nodes in the layer k. This design ensures that a node

has a strong connection to the next layer if it is very similar to other nodes in the current layer. As a consequence, it is likely to guide the random walk to the next layer to acquire more information.

Biased Random Walks on the Built Graph

A biased random walk algorithm is proposed to generate a set of random walks, which are used to generate co-occurrence tuples to be reconstructed. Assume that the random walk is now at the node u in the layer k, for the next step, the random walk stays at the same layer with the probability q and jumps to another layer with the probability $1 - q$, where q is a hyper-parameter.

If the random walk stays at the same layer, the probability of stepping from the current node u to another node v is computed as follows

$$p_k(v|u) = \frac{\exp(-g_k(v,u))}{Z_k(u)},$$

where $Z_k(u)$ is a normalization factor for the node u in the layer k, which is defined as follows:

$$Z_k(u) = \sum_{(v,u)\in\mathcal{E}_k} \exp(-g_k(v,u)).$$

If the walk decides to walk to another layer, the probabilities to the layer $k+1$ and to the layer $k-1$ are calculated as follows:

$$\begin{aligned} p_k\left(u^{(k)}, u^{(k+1)}\right) &= \frac{w\left(u^{(k)},u^{(k+1)}\right)}{w\left(u^{(k)},u^{(k+1)}\right)+w\left(u^{(k)},u^{(k-1)}\right)} \\ p_k\left(u^{(k)}, u^{(k-1)}\right) &= 1 - p_k\left(u^{(k)}, u^{(k+1)}\right) \end{aligned}$$

We can use this biased random walk to generate the set of random walks where we can extract the co-occurrence relations between nodes. Note that the co-occurrence relations are only extracted between different nodes, but not between the same node from different layers. In other words, the co-occurrence relations are only generated when the random walk takes steps with the same layer. These co-occurrence relations can serve as the information to be reconstructed from the embedding domain as DeepWalk.

4.2.3 Preserving Node Status

Global status of nodes, such as their centrality scores introduced in Section 2.3.3, is one type of important information in graphs. In (Ma et al., 2017), a graph embedding method is proposed to preserve node co-occurrence information and node global status jointly. The method mainly consists of two components: 1) a component to preserve the co-occurrence information; and 2) a component

to keep the global status. The component to preserve the co-occurrence information is the same as Deepwalk that is introduced in Section 4.2.1. Hence, in this section, we focus on the component to preserve the global status information. Instead of preserving global status scores for nodes in the graph, the proposed method aims to preserve their global status ranking. Hence, the extractor calculates the global status scores and then ranks the nodes according to their scores. The reconstructor is utilized to restore the ranking information. Next, we detail the extractor and the reconstructor.

Extractor

The extractor first calculates the global status scores and then obtains the global rank of the nodes. Any of the centrality measurements introduced in Section 2.3.3 can be utilized to calculate the global status scores. After obtaining the global status scores, the nodes can be rearranged in descending order according to the scores. We denote the rearranged nodes as $(v_{(1)}, \ldots, v_{(N)})$ where the subscript indicate the rank of the node.

Reconstructor

The reconstructor is to recover the ranking information extracted by the extractor from the node embeddings. To reconstruct the global ranking, the reconstructor in (Ma et al., 2017) aims to preserve relative ranking of all pairs of nodes in $(v_{(1)}, \ldots, v_{(N)})$. Assume that the order between a pair of nodes is independent of other pairs in $(v_{(1)}, \ldots, v_{(N)})$, then the probability of the global ranking preserved can be modeled by using the node embedding as:

$$p_{global} = \prod_{1 \le i < j \le N} p(v_{(i)}, v_{(j)}),$$

where $p(v_{(i)}, v_{(j)})$ is the probability that node $v_{(i)}$ is ranked before $v_{(j)}$ based on their node embeddings. In detail, it is modeled as:

$$p\left(v_{(i),}, v_{(j)}\right) = \sigma\left(\mathbf{w}^T(\mathbf{u}_{(i)} - \mathbf{u}_{(j)})\right),$$

where $\mathbf{u}_{(i)}$ and $\mathbf{u}_{(j)}$ are the node embeddings for nodes $v_{(i)}$ and $v_{(j)}$ respectively (or outputs of the mapping function for $v_{(i)}$ and $v_{(j)}$), and $\mathbf{w}$ is a vector of parameters to be learned. To preserve the order information, we expect that any ordered pair $(v_{(i)}, v_{(j)})$ should have a high probability to be constructed from the embedding. This can be achieved by minimizing the following objective function:

$$\mathcal{L}_{global} = -\log p_{global}.$$

Note that this objective $\mathcal{L}_{global}$ can be combined with the objective to preserve the co-occurrence information such that the learned embeddings can preserve both the co-occurrence information and the global status.

4.2.4 Preserving Community Structure

Community structure is one of the most prominent features in graphs (Newman, 2006) that has motivated the development of embedding methods to preserve such critical information (Wang et al., 2017c; Li et al., 2018d). A matrix factorization based method is proposed to preserve both node-oriented structure, such as connections and co-occurrence, and community structure (Wang et al., 2017c). Next, we first use the general framework to describe its component to preserve node-oriented structure information, then introduce the component to preserve the community structure information with modularity maximization and finally discuss its overall objective.

Preserving Node-oriented Structure

Two types of node-oriented structure information are preserved (Wang et al., 2017c) – one is pairwise connectivity information, and the other is the similarity between the neighborhoods of nodes. Both types of information can be extracted from the given graph and represented in the form of matrices.

Extractor

The pairwise connection information can be extracted from the graph and be represented as the adjacency matrix $\mathbf{A}$. The goal of the reconstrcutor is to reconstruct the pairwise connection information (or the adjacency matrix) of the graph. The neighborhood similarity measures how similar the neighborhoods of two nodes are. For nodes v_i and v_j, their pairwise neighborhood similarity is computed as follows:

$$s_{i,j} = \frac{\mathbf{A}_i \mathbf{A}_j{}^\top}{\|\mathbf{A}_i\|\|\mathbf{A}_j\|},$$

where $\mathbf{A}_i$ is the i-th row of the adjacency matrix, which denotes the neighborhood information of the node v_i. $s_{i,j}$ is larger when nodes v_i and v_j share more common neighbors and it is 0 if v_i and v_j do not share any neighbors. Intuitively if v_i and v_j share many common neighbors, i.e., $s_{i,j}$ is large, they are likely to co-occur in the random walks described in DeepWalk. Hence, this information has an implicit connection with the co-occurrence. These pairwise neighborhood similarity relations can be summarized in a matrix $\mathbf{S}$, where the i, j-th element is $s_{i,j}$. In summary, the extracted information can be denoted by

two matrices $\mathbf{A}$ and $\mathbf{S}$.

Reconstructor and Objective

The reconstructor aims to recover these two types of extracted information in the form of $\mathbf{A}$ and $\mathbf{S}$. To reconstruct them simultaneously, it first linearly combines them as follows:

$$\mathbf{P} = \mathbf{A} + \eta \cdot \mathbf{S},$$

where $\eta > 0$ controls the importance of the neighborhood similarity. Then, the matrix $\mathbf{P}$ is reconstructed from the embedding domain as : $\mathbf{W}_{con}\mathbf{W}_{cen}^T$, where $\mathbf{W}_{con}$ and $\mathbf{W}_{cen}$ are the parameters of two mapping functions f_{con} and f_{cen}. They have the same design as DeepWalk. The objective can be formulated as follows:

$$\mathcal{L}(\mathbf{W}_{con}, \mathbf{W}_{cen}) = \|\mathbf{P} - \mathbf{W}_{con}\mathbf{W}_{cen}^T\|_F^2,$$

where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix.

Preserving the Community Structure

One popular community detection method is based on modularity maximization (Newman, 2006). Specifically, for a graph with 2 communities, the modularity is defined as:

$$Q = \frac{1}{2 \cdot \text{vol}(\mathcal{G})} \sum_{ij} (\mathbf{A}_{i,j} - \frac{d(v_i)d(v_j)}{\text{vol}(\mathcal{G})}) h_i h_j,$$

where $d(v_i)$ is the degree of node v_i, $h_i = 1$ if node v_i belongs to the first community, otherwise, $h_i = -1$ and $\text{vol}(\mathcal{G}) = \sum_{v_i \in \mathcal{V}} d(v_i)$. In fact, $\frac{d(v_i)d(v_j)}{vol(\mathcal{G})}$ is the expected number of edges between nodes v_i and v_j in a randomly generated graph where the same number of edges as $\mathcal{G}$ are randomly placed. Therefore, the modularity Q measures the difference between the number of edges within communities of a given graph and a randomly generated graph with the same number of edges. Ideally, a good community assignment will result in a large value of Q. Hence, the community can be detected by maximizing the modularity Q. Furthermore, the modularity Q can be defined in a matrix form as :

$$Q = \frac{1}{2 \cdot \text{vol}(\mathcal{G})} \mathbf{h}^T \mathbf{B} \mathbf{h},$$

where $\mathbf{h} \in \{-1, 1\}^N$ is the community membership indicator vector with the i-th element $\mathbf{h}[i] = h_i$ and $\mathbf{B} \in \mathbb{R}^{N \times N}$ is defined as :

$$\mathbf{B}_{i,j} = \mathbf{A}_{i,j} - \frac{d(v_i)d(v_j)}{\text{vol}(\mathcal{G})}.$$

The definition of the modularity can be extended to $m > 2$ communities. In detail, the community membership indicator $\mathbf{h}$ can be generalized as a matrix $\mathbf{H} \in \{0, 1\}^{N \times m}$ where each column of $\mathbf{H}$ represents a community. The i-th row of the matrix $\mathbf{H}$ is a one-hot vector indicating the community of node v_i, where only one element of this row is 1 and others are 0. Therefore, we have $tr(\mathbf{H}^T\mathbf{H}) = N$, where $tr(\mathbf{X})$ denotes the trace of a matrix $\mathbf{X}$. After discarding some constants, the modularity for a graph with m communities can be defined as: $Q = tr(\mathbf{H}^T\mathbf{B}\mathbf{H})$. The assignment matrix $\mathbf{H}$ can be learned by maximizing the modularity Q as:

$$\max_{\mathbf{H}} Q = tr(\mathbf{H}^T\mathbf{B}\mathbf{H}), \quad s.t.\ tr(\mathbf{H}^T\mathbf{H}) = N.$$

Note that $\mathbf{H}$ is a discrete matrix which is often relaxed to be a continuous matrix during the optimization process.

The Overall Objective

To jointly preserve the node-oriented structure information and the community structure information, another matrix $\mathbf{C}$ is introduced to reconstruct the indicator matrix $\mathbf{H}$ together with $\mathbf{W}_{cen}$. As a result, the objective of the entire framework is as:

$$\min_{\mathbf{W}_{con}, \mathbf{W}_{cen}, \mathbf{H}, \mathbf{C}} \|\mathbf{P} - \mathbf{W}_{con}\mathbf{W}_{cen}^T\|_F^2 + \alpha\|\mathbf{H} - \mathbf{W}_{cen}\mathbf{C}^T\|_F^2 - \beta \cdot tr(\mathbf{H}^T\mathbf{B}\mathbf{H}),$$
$$s.t. \quad \mathbf{W}_{con} \geq 0, \mathbf{W}_{cen} \geq 0, \mathbf{C} \geq 0, tr(\mathbf{H}^T\mathbf{H}) = N.$$

where the term $\|\mathbf{H} - \mathbf{W}_{cen}\mathbf{C}^T\|_F^2$ connects the community structure information with the node representations, the non-negative constraints are added as non-negative matrix factorization is adopted by (Wang et al., 2017c) and the hyperparameters α and β control the balance among three terms.

4.3 Graph Embedding on Complex Graphs

In previous sections, we have discussed graph embedding algorithms for simple graphs. However, as shown in Section 2.6, real-world graphs present much more complicated patterns, resulting in numerous types of complex graphs. In this section, we introduce embedding methods for these complex graphs.

4.3.1 Heterogeneous Graph Embedding

In heterogeneous graphs, there are different types of nodes. In (Chang et al., 2015), a framework HNE was proposed to project different types of nodes in the heterogeneous graph into a common embedding space. To achieve this goal, a distinct mapping function is adopted for each type. Nodes are assumed to be associated with node features that can have different forms (e.g., images or texts) and dimensions. Thus, different deep models are employed for each type of nodes to map the corresponding features into the common embedding space. For example, if the associated feature is in the form of images, CNNs are adopted as the mapping function. HNE aims to preserve the pairwise connections between nodes. Thus, the extractor in HNE extracts node pairs with edges as the information to be reconstructed, which can be naturally denoted by the adjacency matrix $\mathbf{A}$. Hence, the reconstructor is to recover the adjacency matrix $\mathbf{A}$ from the node embeddings. Specifically, given a pair of nodes (v_i, v_j) and their embeddings $\mathbf{u}_i, \mathbf{u}_j$ learned by the mapping functions, the probability of the reconstructed adjacency element $\tilde{\mathbf{A}}_{i,j} = 1$ is computed as follows:

$$p(\tilde{\mathbf{A}}_{i,j} = 1) = \sigma(\mathbf{u}_i^\top \mathbf{u}_j),$$

where σ is the sigmoid function. Correspondingly,

$$p(\tilde{\mathbf{A}}_{i,j} = 0) = 1 - \sigma(\mathbf{u}_i^\top \mathbf{u}_j).$$

The goal is to maximize the probability such that the reconstructed adjacency matrix $\tilde{\mathbf{A}}$ is close to the original adjacency matrix $\mathbf{A}$. Therefore, the objective is modeled by the cross-entropy as follows:

$$-\sum_{i,j=1}^{N} \Big(\mathbf{A}_{i,j} \log p(\tilde{\mathbf{A}}_{i,j} = 1) + (1 - \mathbf{A}_{i,j}) \log p(\tilde{\mathbf{A}}_{i,j} = 0) \Big). \tag{4.10}$$

The mapping functions can be learned by minimizing the objective in Eq. (4.10) where the embeddings can be obtained. In heterogeneous graphs, different types of nodes and edges carry different semantic meanings. Thus, for heterogeneous network embedding, we should not only care about the structural correlations between nodes but also their semantic correlations. metapath2vec (Dong et al., 2017) is proposed to capture both correlations between nodes. Next, we detail the metapath2vec (Dong et al., 2017) algorithm including its extractor, reconstructor and objective. Note that the mapping function in metapath2vec is the same as DeepWalk.

Meta-path based Information Extractor

To capture both the structural and semantic correlations, meta-path based random walks are introduced to extract the co-occurrence information. Specif-

ically, meta-paths are employed to constrain the decision of random walks. Next, we first introduce the concept of meta-paths and then describe how to design the meta-path based random walk.

Definition 4.5 (Meta-path Schema) Given a heterogeneous graph $\mathcal{G}$ as defined in Definition 2.35, a meta-path schema ψ is a meta-template in $\mathcal{G}$ denoted as $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R2} \cdots \xrightarrow{R_l} A_{l+1}$, where $A_i \in \mathcal{T}_n$ and $R_i \in \mathcal{T}_e$ denote certain types of nodes and edges, respectively. The meta path schema defines a composite relation between nodes from type A_1 to type A_{l+1} where the relation can be denoted as $R = R_1 \circ R_2 \circ \cdots R_{l-l} \circ R_l$. A meta-path instance of type ψ is a path with the meta-path schema ψ, where each node and edge in the path follows the corresponding types in the schema.

Meta-path schema can be used to guide the random walks. A meta-path-based random walk is a randomly generated instance of a given meta-path schema ψ. The formal definition of a meta-path based random walk is given below:

Definition 4.6 Given a meta-path schema $\psi : A_1 \xrightarrow{R_1} A_2 \xrightarrow{R2} \cdots \xrightarrow{R_l} A_{l+1}$, the transition probability of a random walk guided by ψ can be computed as:

$$p(v^{(t+1)}|v^{(t)}, \psi) = \begin{cases} \frac{1}{\left|\mathcal{N}_{t+1}^{R_t}(v^{(t)})\right|}, & \text{if } v^{(t+1)} \in \mathcal{N}_{t+1}^{R_t}(v^{(t)}), \\ 0, & \text{otherwise,} \end{cases}$$

where $v^{(t)}$ is a node of type A_t, corresponding to the position of A_t in the meta-path schema. $\mathcal{N}_{t+1}^{R_t}(v^{(t)})$ denotes the set of neighbors of $v^{(t)}$ which have the node type A_{t+1} and connect to $v^{(t)}$ through edge type R_t. It can be formally defined as:

$$\mathcal{N}_{t+1}^{R_t}(v^{(t)}) = \{v_j \mid v_j \in \mathcal{N}(v^{(t)}) \text{ and } \phi_n(v_j) = A_{t+1} \text{ and } \phi_e(v^{(t)}, v_j) = R_t\}.$$

where $\phi_n(v_j)$ is a function to retrieve the type of node v_j and $\phi_e(v^{(t)}, v_j)$ is a function to retrieve the type of edge $(v^{(t)}, v_j)$ as introduced in Definition 2.35.

Then, we can generate random walks under the guidance of various meta-path schemas from which co-occurrence pairs can be extracted in the same way as that in Section 4.2.1. Likewise, we denote tuples extracted from the random walks in the form of (v_{con}, v_{cen}) as $\mathcal{I}$.

Reconstructor

There are two types of reconstructors proposed in (Dong et al., 2017). The first one is the same as that for DeepWalk (or Eq. (4.2)) in Section 4.2.1. The other reconstorctor is to define a multinomial distribution for each type of nodes

instead of a single distribution over all nodes as Eq. (4.2). For a node v_j with type nt, the probability of observing v_j given v_i can be computed as follows:

$$p(v_j|v_i) = \frac{exp(f_{con}(v_j)^\top f_{cen}(v_i))}{\sum_{v \in \mathcal{V}_{nt}} exp(f_{con}(v) f_{cen}^\top(v_i))},$$

where $\mathcal{V}_{nt}$ is a set consisting of all nodes with type $nt \in \mathcal{T}_n$. We can adopt either of the two reconstructors and then construct the objective in the same way as that of DeepWalk in Section 4.2.1.

4.3.2 Bipartite Graph Embedding

As defined in Definition 2.36, in bipartite graphs, there are two disjoint sets of nodes $\mathcal{V}_1$ and $\mathcal{V}_2$, and no edges are existing within these two sets. For convenience, we use $\mathcal{U}$ and $\mathcal{V}$ to denote these two disjoint sets. In (Gao et al., 2018b), a bipartite graph embedding framework BiNE is proposed to capture the relations between the two sets and the relations within each set. Especially, two types of information are extracted: 1) the set of edges $\mathcal{E}$, which connect the nodes from the two sets, and 2) the co-occurrence information of nodes within each set. The same mapping function as DeepWalk is adopted to map the nodes in the two sets to the node embeddings. We use $\mathbf{u}_i$ and $\mathbf{v}_i$ to denote the embeddings for nodes $u_i \in \mathcal{U}$ and $v_i \in \mathcal{V}$, respectively. Next, we introduce the information extractor, reconstructor and the objective for BiNE.

Information Extractor

Two types of information are extracted from the bipartite graph. One is the edges between the nodes from the two node sets, denoted as $\mathcal{E}$. Each edge $e \in \mathcal{E}$ can be represented as $(u_{(e)}, v_{(e)})$ with $u_{(e)} \in \mathcal{U}$ and $v_{(e)} \in \mathcal{V}$. The other is the co-occurrence information within each node set. To extract the co-occurrence information in each node set, two homogeneous graphs with $\mathcal{U}$ and $\mathcal{V}$ as node sets are induced from the bipartite graph, respectively. Specifically, if two nodes are 2-hop neighbors in the original graph, they are connected in the induced graphs. We use $\mathcal{G}_\mathcal{U}$ and $\mathcal{G}_\mathcal{V}$ to denote the graphs induced for node sets $\mathcal{V}$ and $\mathcal{U}$, respectively. Then, the co-occurrence information can be extracted from the two graphs in the same way as DeepWalk. We denote the extracted co-occurrence information as $\mathcal{I}_\mathcal{U}$ and $\mathcal{I}_\mathcal{V}$, respectively. Therefore, the information to be reconstructed includes the set of edges $\mathcal{E}$ and the co-occurrence information for $\mathcal{U}$ and $\mathcal{V}$.

Reconstructor and Objective

The reconstructor to recover the co-occurrence information in $\mathcal{U}$ and $\mathcal{V}$ from the embeddings is the same as that for DeepWalk. We denote the two objectives for re-constructing $\mathcal{I}_{\mathcal{U}}$ and $\mathcal{I}_{\mathcal{V}}$ as $\mathcal{L}_{\mathcal{U}}$ and $\mathcal{L}_{\mathcal{V}}$, respectively. To recover the set of edges $\mathcal{E}$, we model the probability of observing the edges based on the embeddings. Specifically, given a node pair (u_i, v_j) with $u_i \in \mathcal{U}$ and $v_j \in \mathcal{V}$, we define the probability that there is an edge between the two nodes in the original bipartite graph as:

$$p(u_i, u_j) = \sigma(\mathbf{u}_i^\top \mathbf{v}_j),$$

where σ is the sigmoid function. The goal is to learn the embeddings such that the probability for the node pairs of edges in $\mathcal{E}$ can be maximized. Thus, the objective is defined as

$$\mathcal{L}_{\mathcal{E}} = -\sum_{(u_i, v_j) \in \mathcal{E}} \log p(u_i, v_j).$$

The final objective of BiNE is as follows:

$$\mathcal{L} = \mathcal{L}_{\mathcal{E}} + \eta_1 \mathcal{L}_{\mathcal{U}} + \eta_2 \mathcal{L}_{\mathcal{V}},$$

where η_1 and η_2 are the hyperparameters to balance the contributions for different types of information.

4.3.3 Multi-dimensional Graph Embedding

In a multi-dimensional graph, all dimensions share the same set of nodes, while having their own graph structures. For each node, we aim to learn (1) a general node representation, which captures the information from all the dimensions and (2) a dimension specific representation for each dimension, which focuses more on the corresponding dimension (Ma et al., 2018d). The general representations can be utilized to perform general tasks such as node classification which requires the node information from all dimensions. Meanwhile, the dimension-specific representation can be utilized to perform dimension-specific tasks such as link prediction for a certain dimension. Intuitively, for each node, the general representation and the dimension-specific representation are not independent. Therefore, it is important to model their dependence. To achieve this goal, for each dimension d, we model the dimension-specific representation $\mathbf{u}_{d,i}$ for a given node v_i as

$$\mathbf{u}_{d,i} = \mathbf{u}_i + \mathbf{r}_{d,i}, \tag{4.11}$$

where $\mathbf{u}_i$ is the general representation and $\mathbf{r}_{d,i}$ is the representation capturing information only in the dimension d without considering the dependence. To learn these representations, we aim to reconstruct the co-occurrence relations in different dimensions. Specifically, we optimize mapping functions for $\mathbf{u}_i$ and $\mathbf{r}_{d,i}$ by reconstructing the co-occurrence relations extracted from different dimensions. Next, we introduce the mapping functions, the extractor, the reconstructor and the objective for the multi-dimensional graph embedding.

The mapping functions

The mapping function for the general representation is denoted as $f()$, while the mapping function for a specific dimension d is $f_d()$. Note that all the mapping functions are similar to that in DeepWalk. It is implemented as looking-up tables as follows

$$\mathbf{u}_i = f(v_i) = e_i^\top \mathbf{W},$$
$$\mathbf{r}_{d,i} = f_d(v_i) = e_i^\top \mathbf{W}_d, d = 1 \ldots, D,$$

where D is the number of dimensions in the multi-dimensional graph.

Information Extractor

We extract co-occurrence relations for each dimension d as $\mathcal{I}_d$ using the co-occurrence extractor introduced in Section 4.2.1. The co-occurrence information of all dimensions is the union of that for each dimension as follows:

$$\mathcal{I} = \cup_{d=1}^{D} \mathcal{I}_d.$$

The Reconstructor and Objective

We aim to learn the mapping functions such that the probability of the co-occurrence $\mathcal{I}$ can be well reconstructed. The reconstructor is similar to that in DeepWalk. The only difference is that the reconstructor is now applied to the extracted relations from different dimensions. Correspondingly, the objective can be stated as follows:

$$\min_{\mathbf{W},\mathbf{W}_1,\ldots,\mathbf{W}_D} -\sum_{d=1}^{D} \sum_{(v_{con},v_{cen})\in\mathcal{I}_d} \#(v_{con}, v_{cen}) \cdot \log p(v_{con}|v_{cen}), \tag{4.12}$$

where $\mathbf{W}, \mathbf{W}_1, \ldots, \mathbf{W}_D$ are the parameters of the mapping functions to be learned. Note that in (Ma et al., 2018d), for a given node, the same representation is used for both the center and the context representations.

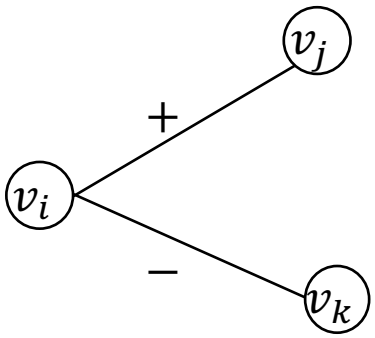


Figure 4.4 A triplet consists of a positive edge and a negative edge

4.3.4 Signed Graph Embedding

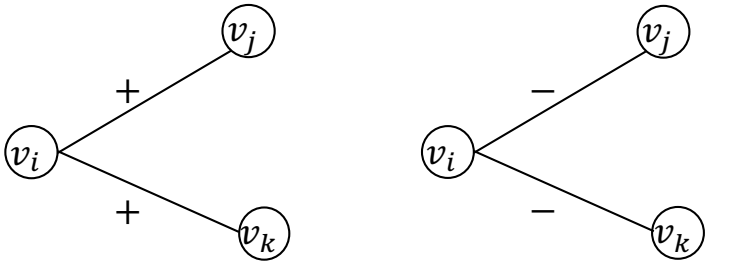
In signed graphs, there are both positive and negative edges between nodes, as introduced in Definition 2.38. Structural balance theory is one of the most important social theories for signed graphs. A signed graph embedding algorithm SiNE based on structural balance theory is proposed (Wang et al., 2017b). As suggested by balance theory (Cygan et al., 2012), nodes should be closer to their "friends" (or nodes with positive edges) than their "foes" (or nodes with negative edges). For example, in Figure 4.4, v_j and v_k can be regarded as the "friend" and "foe" of v_i, respectively. SiNE aims to map "friends" closer than "foes" in the embedding domain, i.e., mapping v_j closer than v_k to v_i. Hence, the information to preserve by SiNE is the relative relations between "friends" and "foes". Note that the mapping function in SiNE is the same as that in DeepWalk. Next, we first describe the information extractor and then introduce the reconstructor.

Information Extractor

The information to preserve can be represented as a triplet (v_i, v_j, v_k) as shown in Figure 4.4, where nodes v_i and v_j are connected by a positive edge while nodes v_i and v_k are connected by a negative edge. Let $\mathcal{I}_1$ denote a set of these triplets in a signed graph, which can be formally defined as:

$$\mathcal{I}_1 = \left\{\left(v_i, v_j, v_k\right) | \mathbf{A}_{i,j} = 1, \ \ \mathbf{A}_{i,k} = -1, \ \ v_i, v_j, v_k \in \mathcal{V}\right\},$$

where $\mathbf{A}$ is the adjacency matrix of the signed graph as defined in Definition 2.38. In the triplet $\left(v_i, v_j, v_k\right)$, the node v_j is supposed to be more similar to v_i than the node v_k according to balance theory. For a given node v, we define its 2-hop subgraph as the subgraph formed by the node v, nodes that are



(a) A triplet with only positive edges (b) A triplet with only negative edges

Figure 4.5 Triplets with edges of the same sign

within 2-hops of v and all the edges between these nodes. In fact, the extracted information $\mathcal{I}_1$ does not contain any information for a node v whose 2-hop subgraph has only positive or negative edges. In this case, all triplets involving v contain edges with the same sign as illustrated in Figure 4.5. Thus, we need to specify the information to preserve for these nodes in order to learn their representations.

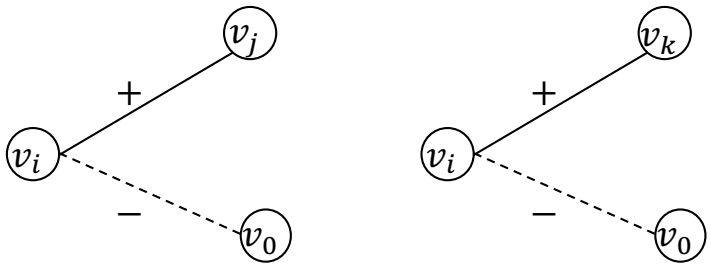
It is evident that the cost of forming negative edges is higher than that of forming positive edges (Tang et al., 2014b). Therefore, in social networks, many nodes have only positive edges in their 2-hop subgraphs while very few have only negative edges in their 2-hop subgraphs. Hence, we only consider to handle nodes whose 2-hop subgraphs have only positive edges, while a similar strategy can be applied to deal with the other type of nodes. To effectively capture the information for these nodes, we introduce a virtual node v_0 and then create negative edges between the virtual node v_0 and each of these nodes. In this way, such triplet (v_i, v_j, v_k) as shown in Figure 4.5a can be split into two triplets (v_i, v_j, v_0) and (v_i, v_k, v_0) as shown in Figure 4.6. Let $\mathcal{I}_0$ denote all these edges involving the virtual node v_0. The information we extract can be denoted as $\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_0$.

Reconstructor

To reconstruct the information of a given triplet, we aim to infer the relative relations of the triplet based on the node embeddings. For a triplet $\left(v_i, v_j, v_k\right)$, the relative relation between v_i, v_j and v_k can be mathematically reconstructed using their embeddings as follows:

$$s\left(f(v_i), f(v_j)\right) - \left(s\left(f(v_i), f(v_k)\right) + \delta\right) \tag{4.13}$$

where $f()$ is the same mapping function as that in Eq.(4.1). The function $s(\cdot, \cdot)$ measures the similarity between two given node representations, which is mod-



(a) (b)

Figure 4.6 Expanding triplet in Figure 4.5a with a virtual node

eled with feedforward neural networks. Eq. (4.13) larger than 0 suggests that v_i is more similar to v_j than v_k. The parameter δ is a threshold to regulate the difference between the two similarities. For example, a lager δ means that v_i and v_j should be much more similar with each other than v_i and v_k to make Eq. (4.13) larger than 0. For any triplet $\left(v_i, v_j, v_k\right)$ in $\mathcal{I}$, we expect Eq.(4.13) to be larger than 0 such that the relative information can be preserved, i.e., v_i and v_j connected with a positive edge are more similar than v_i and v_k connected with a negative edge.

The Objective

To ensure that the information in $\mathcal{I}$ can be preserved by node representations, we need to optimize the mapping function such that Eq.(4.13) can be larger than 0 for all triplets in $\mathcal{I}$. Hence, the objective function can be defined as follows:

$$\begin{aligned}\min_{\mathbf{W},\Theta} \frac{1}{|\mathcal{I}_0| + |\mathcal{I}_1|}[& \sum_{(v_i,v_j,v_k)\in\mathcal{I}_1} \max\left(0, s(f(v_i), f(v_k)) + \delta - s(f(v_i), f(v_j))\right) \\ & + \sum_{(v_i,v_j,v_0)\in\mathcal{I}_0} \max\left((0, s(f(v_i), f(v_0)) + \delta_0 - s(f(v_i), f(v_j))\right) \\ & + \alpha(R(\Theta) + \|\mathbf{W}\|_F^2)]\end{aligned}$$

where $\mathbf{W}$ is the parameters of the mapping function, Θ denotes the parameters of $s(\cdot,\cdot)$ and $R(\Theta)$ is the regularizer on the parameters. Note that, we use different parameters δ and δ_0 for $\mathcal{I}_1$ and $\mathcal{I}_0$ to flexibly distinguish the triplets from the two sources.

4.3.5 Hypergraph Embedding

In a hypergraph, a hyperedge captures relations between a set of nodes, as introduced in Section 2.6.5. In (Tu et al., 2018), a method DHNE is proposed to learn node representations for hypergraphs by utilizing the relations encoded in hyperedges. Specifically, two types of information are extracted from hyperedges that are reconstructed by the embeddings. One is the proximity described directly by hyperedges. The other is the co-occurrence of nodes in hyperedges. Next, we introduce the extractor, the mapping function, the reconstructor and the objective of DHNE.

Information Extractor

Two types of information are extracted from the hypergraph. One is the hyperedges. The set of hyperedges denoted as $\mathcal{E}$ directly describes the relations between nodes. The other type is the hyperedge co-occurrence information. For a pair of nodes v_i and v_j, the frequency they co-occur in hyperedges indicates how strong their relation is. The hyperedge co-occurrence between any pair of nodes can be extracted from the indicate matrix $\mathbf{H}$ as follows:

$$\mathbf{A} = \mathbf{H}\mathbf{H}^\top - \mathbf{D}_v$$

where $\mathbf{H}$ is the indicate matrix and $\mathbf{D}_v$ is the diagonal node degree matrix as introduced in Definition 2.39. The i, j-th element $\mathbf{A}_{i,j}$ indicates the number of times that nodes v_i and v_j co-occurr in hyperedges. For a node v_i, the i-th row of $\mathbf{A}$ describes its co-occurrence information with all nodes in the graph (or the global information of node v_i). In summary, the extracted information includes the set of hyperedges $\mathcal{E}$ and the global co-occurrence information $\mathbf{A}$.

The Mapping Function

The mapping function is modeled with multi-layer feed forward networks with the global co-occurrence information as the input. Specifically, for node v_i, the process can be stated as:

$$\mathbf{u}_i = f(\mathbf{A}_i; \mathbf{\Theta}),$$

where f denotes the feedforward networks with $\mathbf{\Theta}$ as its parameters.

Reconstructor and Objective

There are two reconstructors to recover the two types of extracted information, respectively. We first describe the reconstructor to recover the set of hyperedges $\mathcal{E}$ and then introduce the reconstructor for the co-occurrence information $\mathbf{A}$. To recover the hyperedge information from the embeddings, we

model the probability of a hyperedge existing between any given set of nodes $\{v_{(1)}, \ldots, v_{(k)}\}$ and then aim to maximize the probability for the hyperedge. For convenience, in (Tu et al., 2018), all hyperedges are assumed to have a set of k nodes. The probability that a hyperedge exists in a given set of nodes $\mathcal{V}^i = \{v^i_{(1)}, \ldots, v^i_{(k)}\}$ is defined as:

$$p(1|\mathcal{V}^i) = \sigma\left(g([\mathbf{u}^i_{(1)}, \ldots, \mathbf{u}^i_{(k)}])\right)$$

where $g()$ is a feedforward network that maps the concatenation of the node embeddings to a single scalar and $\sigma()$ is the sigmoid function that transforms the scalar to the probability. Let R^i denote the variable to indicate whether there is a hyperedge between the nodes in $\mathcal{V}^i$ in the hypergraph where $R^i = 1$ denotes that there is an hyperedge while $R^i = 0$ means no hyperedge. Then the objective is modeled based on cross-entropy as:

$$\mathcal{L}_1 = -\sum_{\mathcal{V}^i \in \mathcal{E} \cup \mathcal{E}'} R^i \log p(1|\mathcal{V}^i) + (1 - R^i) \log(1 - p(1|\mathcal{V}^i)),$$

where $\mathcal{E}'$ is a set of negative "hyperedges" that are randomly generated to serve as negative samples. Each of the negative "hyperedge" $\mathcal{V}^i \in \mathcal{E}'$ consists of a set of k randomly sampled nodes.

To recover the global co-occurrence information $\mathbf{A}_i$ for node v_i, a feedforward network, which takes the embedding $\mathbf{u}_i$ as input, is adopted as:

$$\tilde{\mathbf{A}}_i = f_{re}(\mathbf{u}_i; \mathbf{\Theta}_{re}),$$

where $f_{re}()$ is the feedforward network to reconstruct the co-occurrence information with $\mathbf{\Theta}_{re}$ as its parameters. The objective is then defined with least square as:

$$\mathcal{L}_2 = \sum_{v_i \in \mathcal{V}} \|\mathbf{A}_i - \tilde{\mathbf{A}}_i\|_2^2.$$

The two objectives are then combined to form the objective for the entire network embdding framework as:

$$\mathcal{L} = \mathcal{L}_1 + \eta \mathcal{L}_2,$$

where η is a hyperparameter to balance the two objectives.

4.3.6 Dynamic Graph Embedding

In dynamic graphs, edges are associated with timestamps which indicate their emerging time as introduced in Section 2.6.6. It is vital to capture the temporal information when learning the node representations. In (Nguyen et al., 2018),

temporal random walk is proposed to generate random walks that capture temporal information in the graph. The generated temporal random walks are then employed to extract the co-occurrence information to be reconstructed. Since its mapping function, reconstructor and objective are the same as those in DeepWalk, we mainly introduce the temporal random walk and the corresponding information extractor.

Information Extractor

To capture both the temporal and the graph structural information, the temporal random walk is introduced in (Nguyen et al., 2018). A valid temporal random walk consists of a sequence of nodes connected by edges with non-decreasing time stamps. To formally introduce temporal random walks, we first define the set of temporal neighbors for a node v_i at a given time t as:

Definition 4.7 (Temporal Neighbors) For a node $v_i \in \mathcal{V}$ in a dynamic graph $\mathcal{G}$, its temporal neighbors at time t are those nodes connected with v_i after time t. Formally, it can be expressed as follows:

$$\mathcal{N}_{(t)}(v_i) = \{v_j | (v_i, v_j) \in \mathcal{E} \text{ and } \phi_e((v_i, v_j)) \geq t\}$$

where $\phi_e((v_i, v_j))$ is the temporal mapping function. It maps a given edge to its associated time as defined in Definition 2.40.

The temporal random walks can then be stated as follows:

Definition 4.8 (Temporal Random Walks) Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \phi_e\}$ be a dynamic graph where ϕ_e is the temporal mapping function for edges. We consider a temporal random walk starting from a node $v^{(0)}$ with $(v^{(0)}, v^{(1)})$ as its first edge. Assume that at the k-th step, it just proceeds from node $v^{(k-1)}$ to node $v^{(k)}$ and now we choose the next node from the temporal neighbors $\mathcal{N}_{(\phi_e((v^{(k-1)}, v^{(k)})))}(v^{(k)})$ of node $v^{(k)}$ with the following probability:

$$p(v^{(k+1)}|v^{(k)}) = \begin{cases} pre(v^{(k+1)}) & \text{if } v^{(k+1)} \in \mathcal{N}_{(\phi_e((v^{(k-1)}, v^{(k)})))}(v^{(k)}) \\ 0, & \text{otherwise,} \end{cases}$$

where $pre(v^{(k+1)})$ is defined below where nodes with smaller time gaps to the current time are chosen with higher probability:

$$pre(v^{(k+1)}) = \frac{\exp\left[\phi_e((v^{(k-1)}, v^{(k)})) - \phi_e((v^{(k)}, v^{(k+1)}))\right]}{\sum_{v^{(j)} \in \mathcal{N}_{(\phi_e((v^{(k-1)}, v^{(k)})))}(v^{(k)})} \exp\left[\phi_e((v^{(k-1)}, v^{(k)})) - \phi_e((v^{(k)}, v^{(j)}))\right]}.$$

A temporal random walk naturally terminates itself if there are no temporal neighbors to proceed. Hence, instead of generating random walks of fixed

length as DeepWalk, we generate temporal random walks with length between the window size w for co-occurrence extraction and a pre-defined length T. These random walks are leveraged to generate the co-occurrence pairs, which are reconstructed with the same reconstructor as DeepWalk.

4.4 Conclusion

In this chapter, we introduce a general framework and a new perspective to understand graph embedding methods in a unified way. It mainly consists of four components including: 1) a mapping function, which maps nodes in a given graph to their embeddings in the embedding domain; 2) an information extractor, which extracts information from the graphs; 3) a reconstructor, which utilizes the node embeddings to reconstruct the extracted information; and 4) an objective, which often measures the difference between the extracted and reconstructed information. The embeddings can be learned by optimizing the objective. Following the general framework, we categorize graph embedding methods according to the information they aim to preserve including co-occurrence based, structural role based, global status based and community based methods and then detail representative algorithms in each group. Besides, under the general framework, we also introduce representative embedding methods for complex graphs, including heterogeneous graphs, bipartite graphs, multi-dimensional graphs, signed graphs, hypergraphs, and dynamic graphs.

4.5 Further Reading

There are embedding algorithms preserving the information beyond what we have discussed above. In (Rossi et al., 2018), motifs are extracted and are preserved in node representations. A network embedding algorithm to preserve asymmetric transitivity information is proposed in (Ou et al., 2016) for directed graphs. In (Bourigault et al., 2014), node representations are learned to model and predict information diffusion. For complex graphs, we only introduce the most representative algorithms. However, there are more algorithms for each type of complex graphs including heterogeneous graphs (Chen and Sun, 2017; Shi et al., 2018a; Chen et al., 2019b), bipartite graphs (Wang et al., 2019j; He et al., 2019), multi-dimensional graphs (Shi et al., 2018b), signed graphs (Yuan et al., 2017; Wang et al., 2017a), hypergraphs (Baytas et al., 2018) and dynamic graphs (Li et al., 2017a; Zhou et al., 2018b). Besides, there

are quite a few surveys on graph embedding (Hamilton et al., 2017b; Goyal and Ferrara, 2018; Cai et al., 2018; Cui et al., 2018)