A Representation Learning Framework for Property Graphs

Yifan Hou[1], Hongzhi Chen[1], Changji Li[1], James Cheng[1], Ming-Chang Yang[1], Fan Yu[2]
[1]Department of Computer Science and Engineering
The Chinese University of Hong Kong
{yfhou,hzchen,cjli,jcheng,mcyang}@cse.cuhk.edu.hk
[2]Distributed and Parallel Software Lab
Central Software Institute, 2012 Lab, Huawei Technologies Co. Ltd.
fan.yu@huawei.com

ABSTRACT

Representation learning on graphs, also called graph embedding, has demonstrated its significant impact on a series of machine learning applications such as classification, prediction and recommendation. However, existing work has largely ignored the rich information contained in the properties (or attributes) of both nodes and edges of graphs in modern applications, e.g., those represented by property graphs. To date, most existing graph embedding methods either focus on plain graphs with only the graph topology, or consider properties on nodes only. We propose PGE, a graph representation learning framework that incorporates both node and edge properties into the graph embedding procedure. PGE uses node clustering to assign biases to differentiate neighbors of a node and leverages multiple data-driven matrices to aggregate the property information of neighbors sampled based on a biased strategy. PGE adopts the popular inductive model for neighborhood aggregation. We provide detailed analyses on the efficacy of our method and validate the performance of PGE by showing how PGE achieves better embedding results than the state-of-the-art graph embedding methods on benchmark applications such as node classification and link prediction over real-world datasets.

KEYWORDS

graph neural networks, graph embedding, property graphs, representation learning

ACM Reference Format:
Yifan Hou[1], Hongzhi Chen[1], Changji Li[1], James Cheng[1], Ming-Chang Yang[1], Fan Yu[2]. 2019. A Representation Learning Framework for Property Graphs. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3292500.3330948

1 INTRODUCTION

Graphs are ubiquitous today due to the flexibility of using graphs to model data in a wide spectrum of applications. In recent years, more and more machine learning applications conduct classification or prediction based on graph data [7, 15, 17, 28], such as classifying protein's functions in biological graphs, understanding the relationship between users in online social networks, and predicting purchase patterns in buyers-products-sellers graphs in online e-commerce platforms. However, it is not easy to directly make use of the structural information of graphs in these applications as graph data are high-dimensional and non-Euclidean. On the other hand, considering only graph statistics such as degrees [6], kernel functions [14], or local neighborhood structures [24] often provides limited information and hence affects the accuracy of classification/prediction.

Representation learning methods [5] attempt to solve the abovementioned problem by constructing an embedding for each node in a graph, i.e., a mapping from a node to a low-dimensional Euclidean space as vectors, which uses geometric metrics (e.g., Euclidean distance) in the embedding space to represent the structural information. Such graph embeddings [15, 17] have achieved good performance for classification/prediction on *plain graphs* (i.e., graphs with only the pure topology, without node/edge labels and properties). However, in practice, most graphs in real-world do not only contain the topology information, but also contain labels and *properties* (also called *attributes*) on the entities (i.e., nodes) and relationships (i.e., edges). For example, in the companies that we collaborate with, most of their graphs (e.g., various graphs related to products, buyers and sellers from an online e-commerce platform; mobile phone call networks and other communication networks from a service provider) contain rich node properties (e.g., user profile, product details) and edge properties (e.g., transaction records, phone call details). We call such graphs as **property graphs**. Existing methods [10, 16, 18, 22, 30, 31, 36] have not considered to take the rich information carried by both nodes and edges into the graph embedding procedure.

This paper studies the problem of property graph embedding. There are two main challenges. First, each node v may have many properties and it is hard to find which properties may have greater influence on v for a specific application. For example, consider the classification of papers into different topics for a citation graph where nodes represent papers and edges model citation relationships. Suppose that each node has two properties, "year" and "title". Apparently, the property "title" is likely to be more important for paper classification than the property "year". Thus, how to measure the influence of the properties on each node for different applications needs to be considered. Second, for each node v, its neighbors, as well as the connecting edges, may have different properties. How to measure the influences of both the neighbors and the connecting edges on v for different application poses another challenge. In the

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330948

above example, for papers referencing a target paper, those with high citations should mean more to the target paper than those with low citations.

Among existing work, GCN [22] leverages node property information for node embedding generation, while GraphSAGE [18] extends GCN from a spectral method to a spatial one. Given an application, GraphSAGE trains a weight matrix before embedding and then aggregates the property information of the neighbors of each node with the trained matrix to compute the node embedding. However, GraphSAGE does not differentiate neighbors with property dissimilarities for each node, but rather treats all neighbors equally when aggregating their property information. Moreover, GraphSAGE considers only node information and ignores edge directions and properties. Apart from the properties on nodes/edges, real-world graphs also have special structural features. For example, in social networks, nodes are often organized in the form of communities, where similar nodes are either neighbors due to the *homophily* feature [3, 4], or not direct neighbors but with similar structure due to the *structural equivalent* feature [13, 19, 37]. Thus, it is important to also consider structural features. For that, node2vec [16] learns node embeddings by combining two strategies, breadth-first random walk and depth-first random walk, to account for the homophily feature and structural equivalent feature. However, node2vec only utilizes these two structural features without considering any property information.

To address the limitations of existing methods, we propose a new framework, **PGE**, for property graph embedding. PGE applies a biased method to differentiate the influences of the neighbors and the corresponding connecting edges by incorporating both the topology and property information into the graph embedding procedure. The framework consists of three main steps: (1) *property-based node clustering* to classify the neighborhood of a node into similar and dissimilar groups based on their property similarity with the node; (2) *biased neighborhood sampling* to obtain a smaller neighborhood sampled according the bias parameters (which are set based on the clustering result), so that the embedding process can be more scalable; and (3) *neighborhood aggregation* to compute the final low dimensional node embeddings by aggregating the property information of sampled neighborhood with weight matrices trained with neural networks. We also analyze in details how the three steps work together to contribute to a good graph embedding and why our biased method (incorporating node and edge information) can achieve better embedding results than existing methods.

We validated the performance of PGE by comparing with representative graph embedding methods, including DeepWalk [30] and node2vec [16] representing *random walk based methods*, GCN [22] for *graph convolutional networks*, and GraphSAGE [18] for *neighbor aggregation based on weight matrices*. We tested these methods for two benchmark applications, node classification and link prediction, on a variety of real-world graphs. The results show that PGE achieves significant performance improvements over these existing methods. The experimental evaluation validates the importance of incorporating node/edge property information, in addition to topology information, into graph embedding. It also demonstrates the effectiveness of our biased strategy that differentiates neighbors to obtain better embedding results.

2 RELATED WORK

There are three main methods for graph embedding: *matrix factorization*, *random walk*, and *neighbors aggregation*.

For matrix factorization methods, [2, 8] use adjacency matrix to define and measure the similarity among nodes for graph embedding. HOPE [29] further preserves high-order proximities and obtains asymmetric transitivity for directed graphs. Another line of works utilize the random walk statistics to learn embeddings with the skip-gram model [26], which applies vector representation to capture word relationships.

The key idea of random walk is that nodes usually tend to co-occur on short random walks if they have similar embeddings [17]. DeepWalk [30] is the first to input random walk paths into a skip-gram model for learning node embeddings. node2vec [16] further utilizes biased random walks to improve the mapping of nodes to a low-dimensional space, while combining breadth-first walks and depth-first walks to consider graph homophily and structural equivalence. To obtain larger relationships, Walklets [31] involves *offset* to allow longer step length during a random walk, while HARP [10] makes use of graph preprocessing that compresses some nodes into one super-node to improve random walk.

According to [17], *matrix factorization* and *random walk* methods are *shallow embedding approaches* and have the following drawbacks. First, since the node embeddings are independent and there is no sharing of parameters or functions, these methods are not efficient for processing large graphs. Second, they do not consider node/edge properties. Third, as the embeddings are transductive and can only be generated during the training phrase, unseen nodes cannot be embedded with the model being learnt so far.

To address (some of) the above problems, graph-based neural networks have been used to learn node embeddings, which encode nodes into vectors by compressing neighborhood information [9, 20, 36]. However, although this type of methods can share parameters, strictly speaking they are still transductive and have performance bottlenecks for processing large graphs as the input dimensionality of auto-encoders is equal to the number of nodes. Several recent works [11, 18, 22, 23, 34] attempted to use only local neighborhood instead of the entire graph to learn node embeddings through neighbor aggregation, which can also consider property information on nodes. GCN [22] uses graph convolutional networks to learn node embeddings, by merging local graph structures and features of nodes to obtain embeddings from the hidden layers. GraphSAGE [18] is inductive and able to capture embeddings for unseen nodes through its trained auto-encoders directly. The advantage of neighborhood aggregation methods is that they not only consider the topology information, but also compute embeddings by aggregating property vectors of neighbors. However, existing neighborhood aggregation methods treat the property information of neighbors equally and fail to differentiate the influences of neighbors (and their connecting edges) that have different properties.

3 THE PGE FRAMEWORK

We use $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{P}, \mathcal{L}\}$ to denote a property graph , where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. $\mathcal{P}$ is the set of all properties and $\mathcal{P} = \mathcal{P}_{\mathcal{V}} \cup \mathcal{P}_{\mathcal{E}}$, where $\mathcal{P}_{\mathcal{V}} = \bigcup_{v \in \mathcal{V}} \{p_v\}$, $\mathcal{P}_{\mathcal{E}} = \bigcup_{e \in \mathcal{E}} \{p_e\}$, and p_v and p_e are the set of properties of node v and

edge e, respectively. $\mathcal{L} = \mathcal{L}_\mathcal{V} \cup \mathcal{L}_\mathcal{E}$ is the set of labels, where $\mathcal{L}_\mathcal{V}$ and $\mathcal{L}_\mathcal{E}$ are the sets of node and edge labels, respectively. We use $\mathcal{N}_v$ to denote the set of neighbors of node $v \in \mathcal{V}$, i.e., $\mathcal{N}_v = \{v' : (v, v') \in \mathcal{E}\}$. In the case that $\mathcal{G}$ is directed, we may further define $\mathcal{N}_v$ as the set of in-neighbors and the set of out-neighbors, though in this paper we abuse the notation a bit and do not use new notations such as $\mathcal{N}_v^{in}$ and $\mathcal{N}_v^{out}$ for simplicity of presentation, as the meaning should be clear from the context.

The property graph model is general and can represent other popular graph models. If we set $\mathcal{P} = \emptyset$ and $\mathcal{L} = \emptyset$, then $\mathcal{G}$ becomes a *plain graph*, i.e., a graph with only the topology. If we set $\mathcal{P}_\mathcal{V} = \mathcal{A}$, $\mathcal{P}_\mathcal{E} = \emptyset$, and $\mathcal{L} = \emptyset$, where $\mathcal{A}$ is the set of node attributes, then $\mathcal{G}$ becomes an *attributed graph*. If we set $\mathcal{L} = \mathcal{L}_\mathcal{V}$, $\mathcal{P} = \emptyset$, and $\mathcal{L}_\mathcal{E} = \emptyset$, then $\mathcal{G}$ is a *labeled graph*.

3.1 Problem Definition

The main focus of PGE is to utilize both topology and property information in the embedding learning procedure to improve the results for different applications. Given a property graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{P}, \mathcal{L}\}$, we define the similarity between two nodes $v_i, v_j \in \mathcal{V}$ as $s_\mathcal{G}(v_i, v_j)$. The similarity can be further decomposed into two parts, $s_\mathcal{G}(v_i, v_j) = l(s_\mathcal{P}(v_i, v_j), s_\mathcal{T}(v_i, v_j))$, where $s_\mathcal{P}(v_i, v_j)$ is the property similarity and $s_\mathcal{T}(v_i, v_j)$ is the topology similarity between v_i and v_j, and $l(\cdot, \cdot)$ is a non-negative mapping.

The *embedding* of node $v \in \mathcal{V}$ is denoted as $\mathbf{z}_v$, which is a vector obtained by an *encoder* $\text{ENC}(v) = \mathbf{z}_v$. Our objective is to find the optimal $\text{ENC}(\cdot)$, which minimizes the gap $\sum_{v_i,v_j \in \mathcal{V}} ||s_\mathcal{G}(v_i, v_j) - \mathbf{z}_{v_i}^\top \mathbf{z}_{v_j}|| = \sum_{v_i,v_j \in \mathcal{V}} ||l(s_\mathcal{P}(v_i, v_j), s_\mathcal{T}(v_i, v_j)) - \mathbf{z}_{v_i}^\top \mathbf{z}_{v_j}||$.

From the above problem definition, it is apparent that only considering the topology similarity $s_\mathcal{T}(v_i, v_j)$, as the traditional approaches do, cannot converge to globally optimal results. In addition, given a node v and its neighbors v_i, v_j, the property similarity $s_\mathcal{P}(v, v_i)$ can be very different from $s_\mathcal{P}(v, v_j)$. Thus, in the PGE framework, we use both topology similarity and property similarity in learning the node embeddings.

3.2 The Three Steps of PGE

The PGE framework consists of three major steps as follows.

- Step 1: **Property-based Node Clustering.** We cluster nodes in $\mathcal{G}$ based on their properties to produce k clusters $C = \{C_1, C_2, ..., C_k\}$. A standard clustering algorithm such as K-Means [25] or DBSCAN [12] can be used for this purpose, where each node to be clustered is represented by its property vector (note that graph topology information is not considered in this step).
- Step 2: **Biased Neighborhood Sampling.** To combine the influences of property information and graph topology by $l(\cdot, \cdot)$, we conduct biased neighborhood sampling based on the results of clustering in Step 1. To be specific, there are two phases in this step: (1) For each neighbor $v' \in \mathcal{N}_v$, if v' and v are in the same cluster, we assign a *bias* b_s to v' to indicate that they are similar; otherwise we assign a different *bias* b_d to v' instead to indicate that they are dissimilar. (2) We normalize the assigned biases on $\mathcal{N}_v$, and then sample $\mathcal{N}_v$ according to the normalized biases to obtain a fixed-size sampled neighborhood $\mathcal{N}_v^s$.
- Step 3: **Neighborhood Aggregation.** Based on the sampled neighbors $\mathcal{N}_v^s$ in Step 2, we aggregate their property information to obtain $\mathbf{z}_v$ by multiplying the weight matrices that are trained with neural networks.

In the following three sub-sections, we discuss the purposes and details of each of the above three steps.

3.2.1 Property-based Node Clustering. The purpose of Step 1 is to classify $\mathcal{N}_v$ into two types for each node v based on their node property information, i.e., those similar to v or dissimilar to v. If v and its neighbor $v' \in \mathcal{N}_v$ are in the same cluster, we will regard v' as a similar neighbor of v. Otherwise, v' is dissimilar to v.

Due to the high dimensionality and sparsity of properties (e.g., property values are often texts but can also be numbers and other types), which also vary significantly from datasets to datasets, it is not easy to classify the neighborhood of each node into similar and dissimilar groups while maintaining a unified global standard for classifying the neighborhood of all nodes. For example, one might attempt to calculate the property similarity between v and each of v's neighbors, for all $v \in \mathcal{V}$, and then set a threshold to classify the neighbors into similar and dissimilar groups. However, different nodes may require a different threshold and their similarity ranges can be very different. Moreover, each node's neighborhood may be classified differently and as we will show later, the PGE framework actually uses the 2-hop neighborhood while this example only considers the 1-hop neighborhood. Thus, we need a unified global standard for the classification. For this purpose, clustering the nodes based on their properties allows all nodes to be classified based on the same global standard. For example, the 1-hop neighbors and the 2-hop neighbors of a node v are classified in the same way based on whether they are in the same cluster as v.

3.2.2 Biased Neighborhood Sampling. Many real-world graphs have high-degree nodes, i.e., these nodes have a large number of neighbors. It is inefficient and often unnecessary to consider all the neighbors for neighborhood aggregation in Step 3. Therefore, we use the biases b_s and b_d to derive a sampled neighbor set $\mathcal{N}_v^s$ with a fixed size for each node v. As a result, we obtain a sampled graph $\mathcal{G}^s = \{\mathcal{V}, \mathcal{E}^s\}$, where $\mathcal{E}^s = \{(v, v') : v' \in \mathcal{N}_v^s\}$. Since the biases b_s and b_d are assigned to the neighbors based on the clustering results computed from the node properties, $\mathcal{G}^s$ contains the topology information of $\mathcal{G}$ while it is constructed based on the node property information. Thus, Step 2 is essentially a mapping $l(\cdot, \cdot)$ that fuses $s_\mathcal{P}(v, v')$ and $s_\mathcal{T}(v, v')$.

The biases b_s and b_d are the un-normalized possibility of selecting neighbors from dissimilar and similar clusters, respectively. The value of b_s is set to 1, while b_d can be varied depending on the probability (greater b_d means higher probability) that dissimilar neighbors should be selected into $\mathcal{G}^s$. We will analyze the effects of the bias values in Section 4 and verify by experimental results in Section 5.3.2. The size of $\mathcal{N}_v^s$ is set to 25 by default following GraphSAGE [18] (also for fair comparison in our experiments). The size 25 was found to be a good balance point in [18] as a larger size will significantly increase the model computation time, though in the case of PGE as it differentiates neighbors, using a sampled neighborhood could achieve a better quality of embedding than using the full neighborhood.

3.2.3 Neighborhood Aggregation. The last step is to learn the low dimensional embedding with $\mathcal{G}^s = \{\mathcal{V}, \mathcal{E}^s\}$. We use neighborhood

aggregation to learn the function ENC(·) for generating the node embeddings. For each node, we select its neighbors within two hops to obtain $\mathbf{z}_v$ by the following equations:

$$\mathbf{z}_v = \sigma(W^1 \cdot \mathrm{A}(\mathbf{z}^1_v, \sum_{v' \in \mathcal{N}^s_v} \mathbf{z}^1_{v'}/|\mathcal{N}^s_v|)),$$

$$\mathbf{z}^1_{v'} = \sigma(W^2 \cdot \mathrm{A}(p_{v'}, \sum_{v'' \in \mathcal{N}^s_{v'}} p_{v''}/|\mathcal{N}^s_{v'}|)),$$

where p_v is the original property vector of node v, $\sigma(\cdot)$ is the non-linear activation function and $\mathrm{A}(\cdot)$ is the *concatenate* operation. We use two weight matrices W^1 and W^2 to aggregate the node property information of v's one-hop neighbors and two-hop neighbors.

The matrix W^i is used to assign different weights to different properties because aggregating (e.g., taking mean value) node property vectors directly cannot capture the differences between properties, but different properties contribute to the embedding in varying degrees. Also, the weight matrix is data-driven and should be trained separately for different datasets and applications, since nodes in different graphs have different kinds of properties. The weight matrices are pre-trained using Adam SGD optimizer [21], with a loss function defined for the specific application, e.g., for node classification, we use binary cross entropy loss (multi-labeled); for link prediction, we use cross entropy loss with negative sampling.

3.3 Support of Edge Direction and Properties

The sampled graph $\mathcal{G}^s$ does not yet consider the edge direction and edge properties. To include edge properties, we follow the same strategy as we do on nodes. If edges are directed, we consider in-edges and out-edges separately. We cluster the edges into k^e clusters $C^e = \{C^e_1, C^e_2, ..., C^e_{k^e}\}$. Then, we train $2 \cdot k^e$ matrices, $\{W^1_1, W^1_2, ..., W^1_{k^e}\}$ and $\{W^2_1, W^2_2, ..., W^2_{k^e}\}$, to aggregate node properties for k^e types of edges for the 2-hop neighbors. Finally, we obtain $\mathbf{z}_v$ by the following equations:

$$\mathbf{z}_v = \sigma\Big(\mathrm{A}\big(W^1_0 \cdot \mathbf{z}^1_v, \mathrm{A}_{C^e_i \in C^e}(W^1_i \cdot \mathbb{E}_{v' \in \mathcal{N}^s_v \& (v,v') \in C^e_i}[\mathbf{z}^1_{v'}])\big)\Big), \quad (1)$$

$$\mathbf{z}^1_{v'} = \sigma\Big(\mathrm{A}\big(W^2_0 \cdot p_{v'}, \mathrm{A}_{C^e_i \in C^e}(W^2_i \cdot \mathbb{E}_{v'' \in \mathcal{N}^s_{v'} \& (v',v'') \in C^e_i}[p_{v''}])\big)\Big). \quad (2)$$

Note that $|C^e|$ should not be too large as to avoid high-dimensional vector operations. Also, if $|C^e|$ is too large, some clusters may contain only a few elements, leading to under-fitting for the trained weight matrices. Thus, we set $|C^e|$ as a fixed small number.

3.4 The Algorithm

Algorithm 1 presents the overall procedure of computing the embedding vector $\mathbf{z}_v$ of each node $v \in \mathcal{V}$. The algorithm follows exactly the three steps that we have described in Section 3.2.

Algorithm 1 Property Graph Embedding (PGE)

Input: A Property Graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{P}\}$; biases b_d and b_s; the size of sampled neighborhood $|\mathcal{N}^s_v|$; weight matrices $\{W^1_1, W^1_2, ..., W^1_{k^e}\}$ and $\{W^2_1, W^2_2, ..., W^2_{k^e}\}$

Output: Low-dimensional representation vectors $\mathbf{z}_v, \forall v \in \mathcal{V}$

```
Clustering V, E, and obtain C and C^e based on P;                ▷ step 1
for all v ∈ V do                                                 ▷ step 2
    for all v' ∈ N_v do
        Assign b = b_d + (b_s − b_d) · Σ_{C_i ∈ C} I{v, v' ∈ C_i} to v',
        where I{v, v' ∈ C_i} = 1 if v, v' ∈ C_i and 0 otherwise;
    end for
    Sample |N^s_v| neighbors with bias b;
end for
for all v ∈ V do                                                 ▷ step 3
    Compute z^1_v with Equation (2);
end for
for all v ∈ V do
    Compute z_v with Equation (1);
end for
```

4 AN ANALYSIS OF PGE

In this section, we present a detailed analysis of PGE. In particular, we analyze why the biased strategy used in PGE can improve the embedding results. We also discuss how the bias values b_d and b_s and edge information affect the embedding performance.

4.1 The Efficacy of the Biased Strategy

One of the main differences between PGE and GraphSAGE [18] is that neighborhood sampling in PGE is biased (i.e., neighbors are selected based on probability values defined based on b_d and b_s), while GraphSAGE's neighborhood sampling is unbiased (i.e., neighbors are sampled with equal probability). We analyze the difference between the biased and the unbiased strategies in the subsequent discussion.

We first argue that neighborhood sampling is a special case of random walk. For example, if we set the walk length to 1 and perform 10 times of walk, the strategy can be regarded as 1-hop neighborhood sampling with a fixed size of 10. Considering that the random walk process in each step follows an i.i.d. process for all nodes, we define the biased strategy as a $|\mathcal{V}| \times |\mathcal{V}|$ matrix $\mathbf{P}$, where $\mathbf{P}_{i,j}$ is the probability that node v_i selects its neighbor v_j in the random walk. If two nodes v_i and v_j are not connected, then $\mathbf{P}_{i,j} = 0$. Similarly, we define the unbiased strategy $\mathbf{Q}$, where all neighbors of any node have the same probability to be selected. We also assume that there exists an optimal strategy $\mathbf{B}$, which gives the best embedding result for a given application.

A number of works [10, 16, 31] have already shown that adding preference on similar and dissimilar neighbors during random walk can improve the embedding results, based on which we have the following statement: *for a biased strategy $\mathbf{P}$, if $||\mathbf{B}-\mathbf{P}||_1 < ||\mathbf{B}-\mathbf{Q}||_1$, where $\mathbf{B} \neq \mathbf{Q}$, then $\mathbf{P}$ has a positive influence on improving the embedding results.*

Thus, to verify the efficacy of PGE's biased strategy, we need to show that our strategy $\mathbf{P}$ satisfies $||\mathbf{B} - \mathbf{P}||_1 \leq ||\mathbf{B} - \mathbf{Q}||_1$. To do so, we show that b_d and b_s can be used to adjust the strategy $\mathbf{P}$ to get closer to $\mathbf{B}$ (than $\mathbf{Q}$).

Assume that nodes are classified into k clusters $C = \{C_1, C_2, ..., C_k\}$ based on the property information $\mathcal{P}_\mathcal{V}$. For the unbiased strategy, the expected similarity of two nodes $v, v' \in \mathcal{V}$ for each random walk step is:

$$\mathbb{E}[s_{\mathcal{G}(v,v')}] = \frac{\sum_{v \in \mathcal{V}} \sum_{v_i \in \mathcal{N}_v} s_\mathcal{G}(v, v_i)}{|\mathcal{E}|}.$$

The expectation of two nodes' similarity for each walk step in our biased strategy is:

$$\mathbb{E}[s_{\mathcal{G}(v,v')}] = \frac{\sum_{v\in\mathcal{V}}\sum_{v_i\in\mathcal{N}_v\cap C_v} n_s(v)\cdot s_{\mathcal{G}}(v,v_i)}{\frac{|\mathcal{E}|}{k}} + \frac{\sum_{v\in\mathcal{V}}\sum_{v_j\in\mathcal{N}_v\cap(C_v)^c} n_d(v)\cdot s_{\mathcal{G}}(v,v_j)}{\frac{|\mathcal{E}|\cdot(k-1)}{k}}, \quad (3)$$

where $n_s(v)$ and $n_d(v)$ are the normalized biases of b_s and b_d for node v respectively, C_v is the cluster that contains v, and $(C_v)^c = C\backslash\{C_v\}$. Since only connected nodes are to be selected in a random walk step, the normalized biases $n_s(v)$ and $n_d(v)$ can be derived by

$$n_s(v) = \frac{b_s}{b_d\cdot\sum_{v'\in\mathcal{N}_v}\mathbb{I}\{v'\in C_v\} + b_s\cdot\sum_{v'\in\mathcal{N}_v}\mathbb{I}\{v'\in(C_v)^c\}},$$

and

$$n_d(v) = n_s(v)\times\frac{b_d}{b_s}.$$

Consider Equation (3), if we set $b_d = b_s$, which means $n_d(v) = n_s(v)$, then it degenerates to the unbiased random walk strategy. But if we set b_d and b_s differently, we can adjust the biased strategy to either (1) select more dissimilar neighbors by assigning $b_d > b_s$ or (2) select more similar neighbors by assigning $b_s > b_d$.

Assume that the clustering result is not trivial, i.e., we obtain at least more than 1 cluster, we can derive that

$$\frac{\sum_{C_i\in C}\sum_{v,v'\in C_i} s_{\mathcal{P}}(v,v')}{\frac{1}{2}\sum_{C_i\in C}|C_i|\cdot(|C_i|-1)} > \frac{\sum_{v,v'\in\mathcal{V}} s_{\mathcal{P}}(v,v')}{\frac{1}{2}|V|\cdot(|V|-1)}.$$

Since $l(\cdot,\cdot)$ is a non-negative mapping with respect to $s_{\mathcal{P}}(v,v')$, we have

$$\frac{\sum_{C_i\in C}\sum_{v,v'\in C_i} s_{\mathcal{G}}(v,v')}{\frac{1}{2}\sum_{C_i\in C}|C_i|\cdot(|C_i|-1)} > \frac{\sum_{v,v'\in\mathcal{V}} s_{\mathcal{G}}(v,v')}{\frac{1}{2}|V|\cdot(|V|-1)} \quad (4).$$

Equation (4) shows that the similarity $s_{\mathcal{G}}(v,v')$ is higher if v and v' are in the same cluster. Thus, based on Equations (3) and (4), we conclude that parameters b_d and b_s can be used to select similar and dissimilar neighbors.

Next, we consider the optimal strategy $\mathbf{B}$ for 1-hop neighbors, where $\mathbf{B}_{i,j} = \mathbb{I}\{v_j\in\mathcal{N}_{v_i}\}\cdot b^*_{v_i,v_j}$, and $b^*_{v_i,v_j}$ is the normalized optimal bias value for $\mathbf{B}_{i,j}$. Similarly, the unbiased strategy is $\mathbf{Q}_{i,j} = \mathbb{I}\{v_j\in\mathcal{N}_{v_i}\}\cdot\frac{1}{|\mathcal{N}_{v_i}|}$. Thus, we have

$$||\mathbf{B}-\mathbf{Q}||_1 = \sum_{v_i\in\mathcal{V}}\sum_{v_j\in\mathcal{V}}\left|b^*_{v_i,v_j} - \frac{1}{|\mathcal{N}_{v_i}|}\right|.$$

For our biased strategy, $\mathbf{P}_{i,j} = \mathbb{I}\{v_j\in\mathcal{N}_{v_i}\cap C_{v_i}\}\cdot n_s(v) + \mathbb{I}\{v_j\in\mathcal{N}_{v_i}\cap(C_{v_i})^c\}\cdot n_d(v)$. There exist b_s and b_d that satisfy $\sum_{v_i\in\mathcal{V}}\sum_{v_j\in\mathcal{V}}\left|b^*_{v_i,v_j} - \frac{1}{|\mathcal{N}_{v_i}|}\right| \geq \sum_{v_i\in\mathcal{V}}\sum_{v_j\in\mathcal{V}}\left|b^*_{v_i,v_j} - \mathbb{I}\{v_j\in\mathcal{N}_{v_i}\cap C_{v_i}\}\cdot n_s(v) - \mathbb{I}\{v_j\in\mathcal{N}_{v_i}\cap(C_{v_i})^c\}\cdot n_d(v)\right|$, where strict inequality can be derived if $b_d\neq b_s$. Thus, $||\mathbf{B}-\mathbf{P}||_1 < ||\mathbf{B}-\mathbf{Q}||_1$ if we set proper values for b_s and b_d (we discuss the bias values in Section 4.2). Without loss of generality, the above analysis can be extended to the case of multi-hop neighbors.

4.2 The Effects of the Bias Values

Next we discuss how to set the proper values for the biases b_s and b_d for neighborhood sampling. We also analyze the impact of the number of clusters on the performance of PGE.

For neighborhood aggregation in Step 3 of PGE, an accurate embedding of a node v should be obtained by covering the whole connected component that contains v, where all neighbors within k-hops (k is the maximum reachable hop) should be aggregated. However, for a large graph, the execution time of neighborhood aggregation increases rapidly beyond 2 hops, especially for power-law graphs. For this reason, we trade accuracy by considering only the 2-hop neighbors. In order to decrease the accuracy degradation, we can enlarge the change that a neighbor can contribute to the embedding $\mathbf{z}_v$ by selecting dissimilar neighbors within the 2-hops, which we elaborate as follows.

Consider a node $v\in\mathcal{V}$ and its two neighbors $v_i, v_j\in\mathcal{N}_v$, and assume that $\mathcal{N}_{v_i} = \mathcal{N}_{v_j}$ but $|p_v - p_{v_i}| < |p_v - p_{v_j}|$. Thus, we have $s_{\mathcal{T}}(v,v_i) = s_{\mathcal{T}}(v,v_j)$ and $s_{\mathcal{P}}(v,v_i) > s_{\mathcal{P}}(v,v_j)$. Since $l(\cdot,\cdot)$ is a non-negative mapping, we also have $s_{\mathcal{G}}(v,v_i) > s_{\mathcal{G}}(v,v_j)$. Based on the definitions of $\mathbf{z}_v$ and $\mathbf{z}^1_{v'}$ given in Section 3.2.3, by expanding $\mathbf{z}^1_{v'}$ in $\mathbf{z}_v$, we obtain

$$\mathbf{z}_v = \sigma\left(W^1\cdot\mathrm{A}\left(\mathbf{z}^1_v, \sum_{v'\in\mathcal{N}^s_v}\sigma\left(W^2\cdot\mathrm{A}(p_{v'}, \sum_{v''\in\mathcal{N}^s_{v'}} p_{v''}/|\mathcal{N}^s_{v'}|)\right)/|\mathcal{N}^s_v|\right)\right). \quad (5)$$

Equation (5) aggregates the node property vector p_v (which is represented within $\mathbf{z}^1_v$) and the property vectors of v's 2-hop neighbors to obtain the node embedding $\mathbf{z}_v$. This procedure can be understood as transforming from $s_{\mathcal{P}}(v,v')$ to $s_{\mathcal{G}}(v,v')$. Thus, a smaller $s_{\mathcal{P}}(v,v')$ is likely to contribute a more significant change to $\mathbf{z}_v$. With Equation (5), if $|p_v - p_{v_i}| < |p_v - p_{v_j}|$, we obtain $||\mathbf{z}^1_v - \mathbf{z}^1_{v_i}||_1 < ||\mathbf{z}^1_v - \mathbf{z}^1_{v_j}||_1$. Then, for the embeddings, we have $||\mathbf{z}_v - \mathbf{z}_{v_i}||_1 < ||\mathbf{z}_v - \mathbf{z}_{v_j}||_1$. Since v and v_i, as well as v and v_j, have mutual influence on each other, we conclude that for fixed-hop neighborhood aggregation, the neighbors with greater dissimilarity can contribute larger changes to the node embeddings. That is, for fixed-hop neighborhood aggregation, we should set $b_d > b_s$ for better embedding results, which is also validated in our experiments.

Apart from the values of b_d and b_s, the number of clusters obtained in Step 1 of PGE may also affect the quality of the node embeddings. Consider a random graph $\mathcal{G} = \{\mathcal{V},\mathcal{E},\mathcal{P}\}$ with average degree $|\mathcal{E}|/|\mathcal{V}|$. Assume that we obtain k clusters from $\mathcal{G}$ in Step 1, then the average number of neighbors in $\mathcal{N}_v$ that are in the same cluster with a node v is $|\mathcal{N}_v|/k = (|\mathcal{E}|/|\mathcal{V}|)/k$. If k is large, most neighbors will be in different clusters from the cluster of v. On the contrary, a small k means that neighbors in $\mathcal{N}_v$ are more likely to be in the same cluster as v. Neither an extremely large k or small k gives a favorable condition for node embedding based on the biased strategy because we will have either all dissimilar neighbors or all similar neighbors, which essentially renders the neighbors in-differentiable. Therefore, to ensure the efficacy of the biased strategy, the value of k should not fall into either of the two extreme ends. We found that a value of k close to the average degree is a good choice based on our experimental results.

4.3 Incorporating Edge Properties

In addition to the biased values and the clustering number, the edge properties can also bring significant improvements on the embedding results. Many real-world graphs such as online social networks have edge properties like "positive" and "negative". Consider a social network $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{P}\}$ with two types of edges, $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$. Suppose that there is a node $v \in \mathcal{V}$ having two neighbors $v_i, v_j \in \mathcal{N}_v$, and these two neighbors have exactly the same property information $p_{v_i} = p_{v_j}$ and topology information $\mathcal{N}_{v_i} = \mathcal{N}_{v_j}$, but are connected to v with different types of edges, i.e., $(v, v_i) \in \mathcal{E}^+$ and $(v, v_j) \in \mathcal{E}^-$. If we only use Equation (5), then we cannot differentiate the embedding results of v_i and v_j ($\mathbf{z}_{v_i}$ and $\mathbf{z}_{v_j}$). This is because the edges are treated equally and the edge property information is not incorporated into the embedding results. In order to incorporate the edge properties, we introduce an extra matrix for each property. For example, in our case two additional matrices are used for the edge properties "positive" and "negative"; that is, referring to Section 3.3, we have $k^e = 2$ in this case. In the case of directed graphs, we further consider the in/out-neighbors separately with different weight matrices as we have discussed in Section 3.3.

5 EXPERIMENTAL EVALUATION

We evaluated the performance of PGE using two benchmark applications, *node classification* and *link prediction*, which were also used in the evaluation of many existing graph embedding methods [15, 17, 28]. In addition, we also assessed the effects of various parameters on the performance of PGE.

Baseline Methods. We compared PGE with the representative works of the following three methods: *random walk based on skip-gram*, *graph convolutional networks*, and *neighbor aggregation based on weight matrices*.

- DeepWalk [30]: This work introduces the skip-gram model to learn node embeddings by capturing the relationships between nodes based on random walk paths. DeepWalk achieved significant improvements over its former works, especially for multi-labeled classification applications, and was thus selected for comparison.
- node2vec [16]: This method considers both graph homophily and structural equivalence. We compared PGE with node2vec as it is the representative work for graph embedding based on biased random walks.
- GCN [22]: This method is the seminal work that uses convolutional neural networks to learn node embedding.
- GraphSAGE [18]: GraphSAGE is the state-of-the-art graph embedding method and uses node property information in neighbor aggregation. It significantly improves the performance compared with former methods by learning the mapping function rather than embedding directly.

To ensure fair comparison, we used the optimal default parameters of the existing methods. For DeepWalk and node2vec, we used the same parameters to run the algorithms, with window size set to 10, walk length set to 80 and number of walks set to 10. Other parameters were set to their default values. For GCN, GraphSAGE and PGE, the learning rate was set to 0.01. For node classification, we set the epoch number to 100 (for GCN the early stop strategy was used), while for link prediction we set it to 1 (for PubMed we set it to 10 as the graph has a small number of nodes). The other parameters of GCN were set to their optimal default values. PGE also used the same default parameters as those of GraphSAGE such as the number of sampled layers and the number of neighbors.

Table 1: Dataset statistics

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	avg. degree	feature dim.	# of classes
PubMed	19,717	44,338	2.25	500	3
PPI	56,944	818,716	14.38	50	121
BlogCatalog	55,814	1,409,112	25.25	1,000	60
Reddit	232,965	11,606,919	49.82	602	41

Datasets. We used four real-world datasets in our experiments, including a citation network, a biological protein-protein interaction network and two social networks.

- **PubMed** [27] is a set of articles (i.e., nodes) related to diabetes from the PubMed database, and edges here represent the citation relationship. The node properties are TF/IDF-weighted word frequencies and node labels are the types of diabetes addressed in the articles.
- **PPI** [35] is composed of 24 protein-protein interaction graphs, where each graph represents a human tissue. Nodes here are proteins and edges are their interactions. The node properties include positional gene sets, motif gene sets and immunological signatures. The node labels are gene ontology sets. We used the processed version of [18].
- **BlogCatalog** [1] is a social network where users select categories for registration. Nodes are bloggers and edges are relationships between them (e.g., friends). Node properties contain user names, ids, blogs and blog categories. Node labels are user tags.
- **Reddit** [18] is an online discussion forum. The graph was constructed from Reddit posts. Nodes here are posts and they are connected if the same users commented on them. Property information includes the post title, comments and scores. Node labels represent the community. We used the sparse version processed in [18].

Table 1 shows some statistics of the datasets. To evaluate the performance of node classification of the algorithms on each dataset, the labels attached to nodes are treated as classes, whose number is shown in the last column. Note that each node in PPI and BlogCatalog may have multiple labels, while that in PubMed and Reddit has only a single label. The average degree (i.e., $|\mathcal{E}|/|\mathcal{V}|$) shows that the citation dataset PubMed is a sparse graph, while the other graphs have higher average degree. For undirected graphs, each edge is stored as two directed edges.

5.1 Node Classification

We first report the results for node classification. All nodes in a graph were divided into three types: training set, validation set and test set for evaluation. We used 70% for training, 10% for validation and 20% for test for all datasets except for PPI, which is composed of 24 subgraphs and we followed GraphSAGE [18] to use about 80% of the nodes (i.e., those in 22 subgraphs) for training

Table 2: Performance of node classification

F1-Micro (%) Alg. \ Datasets	PubMed	PPI	BlogCatalog	Reddit
DeepWalk	78.85	60.66	38.69	-
node2vec	78.53	61.98	37.79	-
GCN	84.61	-	-	-
GraphSAGE	88.08	63.41	47.22	94.93
PGE	**88.36**	**84.31**	**51.31**	**95.62**

F1-Macro (%) Alg. \ Datasets	PubMed	PPI	BlogCatalog	Reddit
DeepWalk	77.41	45.19	23.73	-
node2vec	77.08	48.57	22.94	-
GCN	84.27	-	-	-
GraphSAGE	87.87	51.85	30.65	92.30
PGE	**88.24**	**81.69**	**37.22**	**93.29**

and nodes in the remaining 2 subgraphs for validation and test. For the biases, we used the default values, $b_s = 1$ and $b_d = 1000$, for all the datasets. For the task of node classification, the embedding result (low-dimensional vectors) satisfies $\mathbf{z}_v \in \mathbb{R}^{d_l}$, where d_l is the number of classes as listed in Table 1. The index of the largest value in $\mathbf{z}_v$ is the classification result for single-class datasets. In case of multiple classes, the rounding function was utilized for processing $\mathbf{z}_v$ to obtain the classification results. We used F1-score [33], which is a popular metric for multi-label classification, to evaluate the performance of classification.

Table 2 reports the results, where the left table presents the F1-Micro values and the right table presents the F1-Macro values. PGE achieves higher F1-Micro and F1-Macro scores than all the other methods for all datasets, especially for PPI and BlogCatalog for which the performance improvements are significant. In general, the methods that use node property information (i.e., PGE, GraphSAGE and GCN) achieve higher scores than the methods that use the skip-gram model to capture the structure relationships (i.e., DeepWalk and node2vec). This is because richer property information is used by the former methods than the latter methods that use only the pure graph topology. Compared with GraphSAGE and GCN, PGE further improves the classification accuracy by introducing biases to differentiate neighbors for neighborhood aggregation, which validates our analysis on the importance of our biased strategy in Section 4. In the remainder of this subsection, we discuss in greater details the performance of the methods on each dataset.

To classify the article categories in PubMed, since the number of nodes in this graph is not large, we used the DBSCAN clustering method in Step 1, which produced $k = 4$ clusters. Note that the graph has a low average degree of only 2.25. Thus, differentiating the neighbors does not bring significant positive influence. Consequently, PGE's F1-scores are not significantly higher than those of GraphSAGE for this dataset.

To classify proteins' functions of PPI, since this graph is not very large, we also used DBSCAN for clustering, which produced $k = 39$ clusters. For this dataset, the improvement made by PGE over other methods is impressive, which could be explained by that neighbors in a protein-protein interaction graph play quite different roles and thus differentiating them may bring significantly benefits for node classification. In fact, although GraphSAGE also uses node property information, since GraphSAGE does not differentiate neighbors, it does not obtain significant improvement over DeepWalk and node2vec (which use structural information only). The small improvement made by GraphSAGE compared with the big improvement made by PGE demonstrates the effectiveness of our biased neighborhood sampling strategy. For GCN, since it does not consider multi-labeled classification, comparing it with the other methods is unfair and not meaningful for this dataset (also for BlogCatalog).

Table 3: Performance of link prediction

MRR (%) Alg. \ Datasets	PubMed	PPI	BlogCatalog	Reddit
GraphSAGE	43.72	39.93	24.61	41.27
PGE (no edge info)	41.47	59.73	23.89	39.81
PGE	**70.77**	**89.21**	**72.97**	**56.59**

BlogCatalog has high feature dimensionality. The original BlogCatalog dataset regards the multi-hot vectors as the feature vectors (with 5,413 dimensions). We used Truncate-SVD to obtain the low-dimensional feature vectors (with 1,000 dimensions). Since the number of nodes is not large, we used DBSCAN for Step 1, which produced $k = 18$ clusters for this dataset. The improvement in the classification accuracy made by PGE is very significant compared with DeepWalk and node2vec, showing the importance of using property information for graph embedding. The improvement over GraphSAGE is also quite significant for this dataset, which is due to both neighbor differentiation and the use of edge direction.

The Reddit graph is much larger than the other graphs, and thus we used K-Means (with $k = 40$) for clustering Reddit instead of using DBSCAN which is much slower. We do not report the results for DeepWalk and node2vec as their training processes did not finish in 10 hours while GraphSAGE and PGE finished in several minutes. We also do not report GCN since it needs to load the full graph matrix into each GPU and ran out of memory on our GPUs (each with 12GB memory). PGE's F1-scores are about 1% higher than those of GraphSAGE, which we believe is a significant improvement given that the accuracy of GraphSAGE is already very high (94.93% and 92.30%).

5.2 Link Prediction

Next we evaluate the quality of graph embedding for link prediction. Given two nodes' embeddings $\mathbf{z}_v$ and $\mathbf{z}_{v'}$, the model should predict whether there is a potential edge existing between them. We used MRR (mean reciprocal rank) [32] to evaluate the performance of link prediction. Specifically, for a node v and $|Q|$ sets of nodes to be predicted, the MRR score can be calculated by the set of prediction queries/lists in Q with $\frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$, where $rank_i$ is the place of the first correct prediction. We compared PGE with GraphSAGE as we did not find the evaluation method for link prediction in DeepWalk, node2vec and GCN. For the sparse citation graph PubMed, we set the epoch number to 10 to avoid the data insufficiency problem. For other datasets, the epoch number was

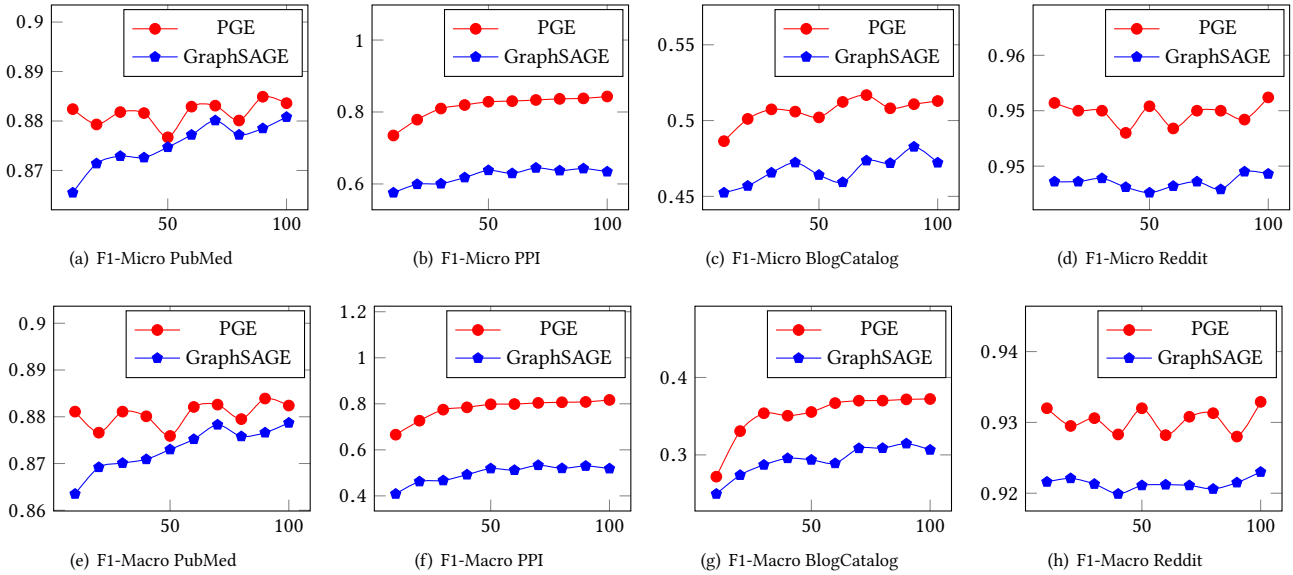

Figure 1: The effects of epoch number

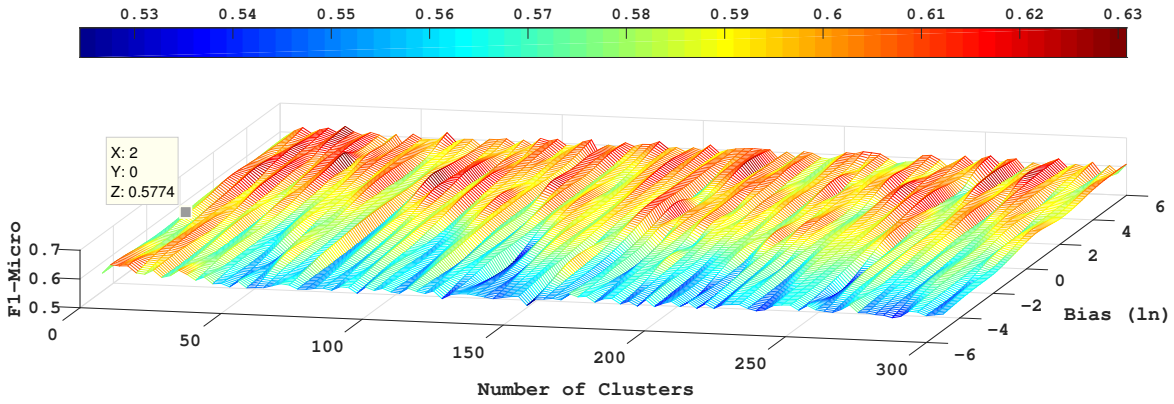

Figure 2: The effects of bias values and cluster number (best viewed as 2D color images)

set to 1. As for the biases b_d and b_s and the clustering methods, they are the same as in the node classification experiment in Section 5.1.

Table 3 reports the MRR scores of PGE and GraphSAGE for the four datasets. We also created a variant of PGE by only considering bias (i.e., the edge information was not used). The results show that without considering the edge information, PGE records lower MRR scores than GraphSAGE except for PPI. However, when the edge information is incorporated, PGE significantly outperforms GraphSAGE in all cases and the MRR score of PGE is at least 37% higher than that of GraphSAGE. According to the MRR score definition, the correct prediction made by PGE is 1 to 3 positions ahead of that made by GraphSAGE. Compared with the improvements made by PGE for node classification, its improvements for link prediction are much more convincing, which can be explained as follows. Differentiating between neighboring nodes may not have a direct effect on predicting a link between two nodes; rather, the use of edge information by PGE makes a significant difference compared with GraphSAGE and the variant of PGE, as the latter two do not use edge information.

5.3 Parameter Sensitivity Tests

In this set of experiments, we evaluated the effects of the parameters in PGE on its performance.

5.3.1 Effects of the Epoch Number. To test the effects of the number of training epochs, we compared PGE with GraphSAGE by varying the epoch number from 10 to 100. We report the F1-Micro and F1-Macro scores for node classification on the four datasets in Figure 1. The results show that PGE and GraphSAGE have similar trends in F1-Micro and F1-Marco, although PGE always outperforms GraphSAGE. Note that the training time increases linearly with the epoch

number, but the training time for 100 epochs is also only tens of seconds (for the small dataset PubMed) to less than 5 minutes (for the largest dataset Reddit).

5.3.2 Effects of Biases and Cluster Number. We also tested the effects of different bias values and the number of clusters. We ran PGE for 1,000 times for node classification on PPI, using different number of clusters k and different values of b_d (by fixing $b_s = 1$). We used K-Means for Step 1 since it is flexible to change the value k. The number of training epochs was set at 10 for each run of PGE. All the other parameters were set as their default values.

Figure 2 reports the results, where the X-axis shows the number of clusters k, the Y-axis indicates the logarithmic value (with the base e) of b_d, and the Z-axis is the F1-Micro score (F1-Macro score is similar and omitted). The results show that taking a larger bias b_d (i.e., $Y > 0$) can bring positive influence on the F1-score independent of the cluster number k, and the performance increases as a larger b_d is used. When b_d is less than 1, i.e., $b_d < b_s$, it does not improve the performance over uniform neighbor sampling (i.e., $b_d = b_s$ or $Y = 0$). This indicates that selecting a larger number of dissimilar neighbors (as a larger b_d means a higher probability of including dissimilar neighbors into $\mathcal{G}^s$) helps improve the quality of node embedding, which is consistent with our analysis in Section 4.

For the number of clusters k, as the average degree of the PPI graph is 14.38, when the cluster number is more than 50, the F1-score becomes fluctuating to k (i.e., the shape is like waves in Figure 2). This phenomenon is caused by the limitation of the clustering algorithm, since K-Means is sensitive to noises and a large k is more likely to be affected by noises. Note that when the cluster number is not large (less than 50), a small bias b_d (less than 1) may also improve the F1-score, which may be explained by the fact that there are homophily and structural equivalence features in the graph, while $b_d < 1$ indicates that nodes tend to select similar neighbors to aggregate. In general, however, a large b_d and a small cluster number k (close to the average degree) are more likely to improve the performance of the neighborhood aggregation method.

6 CONCLUSIONS

We presented a representation learning framework, called PGE, for property graph embedding. The key idea of PGE is a three-step procedure to leverage both the topology and property information to obtain a better node embedding result. Our experimental results validated that, by incorporating the richer information contained in a property graph into the embedding procedure, PGE achieves better performance than existing graph embedding methods such as DeepWalk [30], node2vec [16], GCN [22] and GraphSAGE [18]. PGE is a key component in the GNN library of MindSpore — a unified training and inference framework for device, edge, and cloud in Huawei's full-stack, all-scenario AI portfolio — and has a broad range of applications such as recommendation in Huawei's mobile services, cloud services and 5G IoT applications.

REFERENCES

[1] Nitin Agarwal, Huan Liu, Sudheendra Murthy, Arunabha Sen, and Xufei Wang. A social identity approach to identify familiar strangers in a social network. In *ICWSM*, 2009.

[2] Amr Ahmed, Nino Shervashidze, Shravan M. Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. Distributed large-scale natural graph factorization. In *WWW*, pages 37–48, 2013.

[3] Eytan Bakshy, Itamar Rosenn, Cameron Marlow, and Lada A. Adamic. The role of social networks in information diffusion. In *WWW*, pages 519–528, 2012.

[4] Mauro Barone and Michele Coscia. Birds of A feather scam together: Trustworthiness homophily in A business network. *Social Networks*, 54.

[5] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *PAMI*, 35:1798–1828, 2013.

[6] Smriti Bhagat, Graham Cormode, and S. Muthukrishnan. Node classification in social networks. In *Social network datanalytics*, pages 115–148. 2011.

[7] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34.

[8] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *CIKM*, pages 891–900, 2015.

[9] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152, 2016.

[10] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. HARP: hierarchical representation learning for networks. In *AAAI*, pages 2127–2134, 2018.

[11] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *ICML*, pages 941–949, 2018.

[12] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*, pages 226–231, 1996.

[13] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, 2010.

[14] Thomas Gärtner, Tamás Horváth, and Stefan Wrobel. Graph kernels. In *Encyclopedia of Machine Learning and Data Mining*, pages 579–581. 2017.

[15] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *KBS*, 151:78–94, 2018.

[16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864, 2016.

[17] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 40:52–74, 2017.

[18] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.

[19] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. Rolx: Structural role extraction & mining in large graphs. In *SIGKDD*, pages 1231–1239, 2012.

[20] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[22] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, 2016.

[23] Thomas N. Kipf and Max Welling. Variational graph auto-encoders. *CoRR*, 2016.

[24] David Liben-Nowell and Jon M. Kleinberg. The link-prediction problem for social networks. *JASIST*, 58:1019–1031, 2007.

[25] James MacQueen. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[27] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. Query-driven active surveying for collective classification. In *MLG*, 2012.

[28] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104.

[29] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *SIGKDD*, pages 1105–1114, 2016.

[30] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *SIGKDD*, pages 701–710, 2014.

[31] Bryan Perozzi, Vivek Kulkarni, and Steven Skiena. Walklets: Multiscale graph embeddings for interpretable network classification. *CoRR*, 2016.

[32] Dragomir R. Radev, Hong Qi, Harris Wu, and Weiguo Fan. Evaluating web-based question answering systems. In *LREC*, 2002.

[33] Yutaka Sasaki. The truth of the f-measure. *Teach Tutor Mater*, 1:1–5, 2007.

[34] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *ESWC*, pages 593–607, 2018.

[35] Chris Stark, Bobby-Joe Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers. Biogrid: A general repository for interaction datasets. *Nucleic Acids Research*, 34:535–539, 2006.

[36] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *SIGKDD*, pages 1225–1234, 2016.

[37] Jaewon Yang and Jure Leskovec. Overlapping communities explain core-periphery organization of networks. *Proceedings of the IEEE*, 102.