TGE-PS: Text-driven Graph Embedding with Pairs Sampling

Liheng Chen[†], **Yanru Qu**[†], **Zhenghui Wang**[†], **Lin Qiu**[†], **Weinan Zhang**[†], **Ken Chen**[‡], **Shaodian Zhang**[‡], **Yong Yu**[†]
[†]Shanghai Jiao Tong University, [‡]Synyi LLC.

Abstract

In graphs with rich text information, constructing expressive graph representations requires incorporating textual information with structural information. Graph embedding models are becoming more and more popular in representing graphs, yet they are faced with two issues: *sampling efficiency* and *text utilization*. Through analyzing existing models, we find their training objectives are composed of pairwise proximities, and there are large amounts of redundant node pairs in Random Walk-based methods. Besides, inferring graph structures directly from texts (also known as zero-shot scenario) is a problem that requires higher text utilization. To solve these problems, we propose a novel Text-driven Graph Embedding with Pairs Sampling (TGE-PS) framework. TGE-PS uses Pairs Sampling (PS) to generate training samples which reduces ∼99% training samples and is competitive compared to Random Walk. TGE-PS uses Text-driven Graph Embedding (TGE) which adopts word- and character-level embeddings to generate node embeddings. We evaluate TGE-PS on several real-world datasets, and experimental results demonstrate that TGE-PS produces state-of-the-art results in traditional and zero-shot link prediction tasks.

Introduction

Graph provides a fundamental tool to represent interconnected entities (e.g., articles, diseases) and their attributes (e.g., entity description). Graphs with rich text information are ubiquitous in many fields, and there is often a strong dependency between graph structure and text structure in these graphs. Figure 1 shows an example. The example graph contains two types of "connections": (i) structural connection between (v_A, v_C), (ii) textual "connections" between (v_C, v_F) and (v_A, v_D). By assuming there is a connection between "bone" and "radius", it is natural to infer the connection between (v_D, v_F). This example shows how textual information exposes structural information. Hence, it is promising to better utilize textual information in graphs.

Graph embedding is famous for its efficient representations for entities in graphs (Goyal and Ferrara 2017; Nishana and Surendran 2013). The most important structures in graphs are interconnections between nodes. A series of models are proposed to maximize edge reconstruction probability with different proximities (Cai, Zheng, and

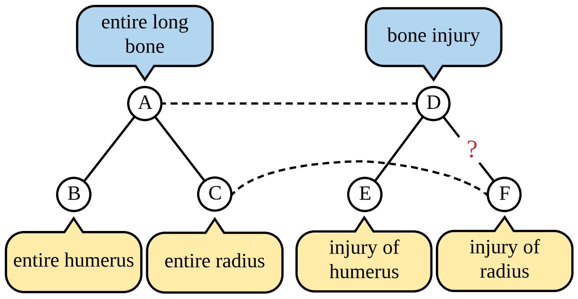

Figure 1: Example graph with informative texts.

Chang 2018), e.g., LINE (Tang et al. 2015), DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) and node2vec (Grover and Leskovec 2016). Although these models are widely used in mapping nodes to low-dimensional dense vectors, they are not well designed for graphs with rich text information. To solve this problem, models like TADW (Yang et al. 2015), CANE (Tu, Liu, and Liu 2017) and Paper2Vec (Ganguly and Pudi 2017) are proposed to utilize textural information with effectiveness in many scenarios.

However, these graph embedding models still need to resolve two issues. The first issue is *sampling efficiency*. The optimization goals of these models can be summarized as maximizing pairwise node similarity, thus the number of training node pairs is critical to training time and can be used as a metric of sampling efficiency. These models usually use edges as training node pairs directly like LINE, or sample training sequences using Random Walk (RW) and extract node pairs within a specific shortest distance like DeepWalk and node2vec. RW generalizes the idea of directly sampling edges and shows better experimental performance under similar settings. However, our theoretical and empirical analysis shows that RW samples redundant node pairs which severely lags efficiency.

The second issue is *text utilization*. Currently, text utlization methods mainly rely on training node embeddings (NE) and text embeddings (TE) together (denoted as NE+TE). NE+TE firstly encodes structural and textual information into an NE space and a TE space separately, and then combines NE and TE together as the final graph representation. However, NE+TE is not suitable for zero-shot settings (See Figure 1), i.e., it is incapable of inferring graph structures directly from texts. Since NE+TE requires both NE and TE, it cannot generate high-quality representations for unseen nodes.

In this paper, we propose a novel Text-driven Graph Embedding with Pairs Sampling (TGE-PS) framework. TGE-PS uses Pairs Sampling (PS) to efficiently generate training samples, and uses Text-driven Graph Embedding (TGE) to produce final node representations. We analyze the redundant sample phenomenon of RW theoretically, and propose PS to improve sampling efficiency. PS samples central-neighbor node pairs directly from central node's neighborhood. We conduct experiments on 7 datasets and PS produces competitive or even better results in link prediction and node classification tasks with much fewer training samples (saving $\sim$99% samples) compared with RW. In embedding stage, we propose TGE method. Since the node embeddings are generated from text embedding in TGE, it can be applied to zero-shot scenarios. The comparison between TGE-PS and other strong baseline models shows that TGE-PS produces remarkably good results in traditional and zero-shot link prediction tasks.

Related Works

Recent years various graph embedding models are becoming more and more popular in representing graph structured data (Goyal and Ferrara 2017; Cai, Zheng, and Chang 2018). These graph embedding models can be categorized into three classes, factorization-based, Random Walk (RW)-based, and deep learning-based. Factorization-based models focus on the adjacent matrices of graphs, and use matrix factorization (MF) to learn low-rank representations of nodes (Ahmed et al. 2013; Cao, Lu, and Xu 2015; Ou et al. 2016; Yang et al. 2015). RW-based models explore the neighborhood of each node through sampling node sequences, and then train node embeddings on the explored neighborhoods. Among RW-based models, DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) and node2vec (Grover and Leskovec 2016) are the most representative, and DeepWalk can be regarded as a special case of node2vec. Deep learning-based models mainly use deep representation learning techniques to improve the quality of node embeddings (Wang, Cui, and Zhu 2016; Cao, Lu, and Xu 2016; Kipf and Welling 2016). There are also other types of graph embedding models like LINE (Tang et al. 2015) and GraphGAN (Wang et al. 2017).

For graphs with rich texts, many models are proposed to incorporate textual information with structural information. TADW (Yang et al. 2015) incorporates text features under the framework of matrix factorization. CANE (Tu, Liu, and Liu 2017) uses convolutional networks and mutual attention mechanism to learn text embeddings, which are interacted with node embeddings via vector inner product. These two models encode structural information and textual information into two separate embedding spaces and generate final node representations from the interactions between these two spaces. Paper2Vec (Ganguly and Pudi 2017) pre-trains node embeddings with text embeddings in Skip-gram model, and then the node embeddings are trained with node2vec. STNE (Liu et al. 2018) "self-translates" sequences of text embeddings into sequences of node embeddings. All the above models rely on known connections to generate graph embeddings, thus are not applicable to zero-shot scenarios.

Text-driven Graph Embedding with Pairs Sampling Framework

Our Text-driven Graph Embedding with Pairs Sampling (TGE-PS) framework includes two stages: sampling and embedding. We propose Pairs Sampling (PS) method for efficient sampling and Text-driven Graph Embedding (TGE) method for better utilizing texts. Inside TGE-PS, TGE benefits from PS for that small quantity of node pairs greatly reduces training time and hence TGE is capable of adopting time-costing yet expressive RNN-based structure. We show the architecture of TGE-PS in Figure 2. For simplicity, we omit the fully-connected layer and the lookup layer in Figure 2. In this section, we firstly give definitions of notions, and then introduce PS and TGE separately.

Definitions

Let $G = (V, E)$ be the given graph and $f : V \to \mathbb{R}^d$ be the mapping function from the node set to the embedding space, where d is the dimension of the embedding space. We denote the central and context embeddings of node v_i as $\mathbf{e}_i = f(v_i)$ and $\mathbf{e}'_i = f'(v_i)$, and the trainable parameters as θ. We define the distance $dist(u, v)$ between u and v as the length of the shortest path between them. And we define node v_i's o-neighborhood $\mathcal{N}_i^o$ as a set of nodes within a given distance o from v_i, $\mathcal{N}^o(v_i) = \{v_j \in V | dist(v_i, v_j) \leq o\}$.

Pairs Sampling Method

Random Walk Since DeepWalk can be regarded as a special case of node2vec, our discussion mainly focuses on node2vec. As stated in (Grover and Leskovec 2016), the objective of node2vec is to maximize the log-probability of observing the o-neighborhood $\mathcal{N}_i^o$ of a node v_i given $\mathbf{e}_i$, $\log \Pr(\mathcal{N}_i^o | v_i)$. node2vec makes conditional independence assumption to simplify the objective as:

$$\max_\theta \sum_{v_i \in V} \sum_{v_j \in \mathcal{N}_i^k} \tau_{i,j} - |\mathcal{N}_i^k| \log Z_i, \tag{1}$$

where Z_i denotes the normalizing term and $\tau_{i,j}$ is the abbreviation of the score function $\tau(v_i, v_j)$ representing conditional probability $p(v_j|v_i)$. Since Z_i is usually approximated by hierarchical softmax or negative sampling in training and relies on $\tau(v_i, v_j)$, we mainly focus on the scoring terms $\sum_i \sum_j \tau_{i,j}$. By making symmetry in feature space assumption, node2vec defines $\tau_{i,j}$ as the inner product of embeddings, i.e., $\mathbf{e}'^{\top}_j \mathbf{e}_i$. The training objective becomes to maximize

$$\begin{aligned} &\sum_{v_i \in V} \sum_{s \in S} \sum_{v_{j'} \in \omega_i^s} \mathbf{e}'^{\top}_{j'} \mathbf{e}_i \\ =& \sum_{v_i \in V} (T + T_i) \Big(\sum_{v_j \in \mathcal{N}_i^k} \frac{T_{j|i}}{2W(T + T_i)} \mathbf{e}'^{\top}_{j'} \mathbf{e}_i \Big) \\ =& T \sum_{v_i \in V} (1 + \alpha_i) \Big(\sum_{v_j \in \mathcal{N}_i^k} \beta_{j|i} \mathbf{e}'^{\top}_{j'} \mathbf{e}_i \Big), \end{aligned} \tag{2}$$

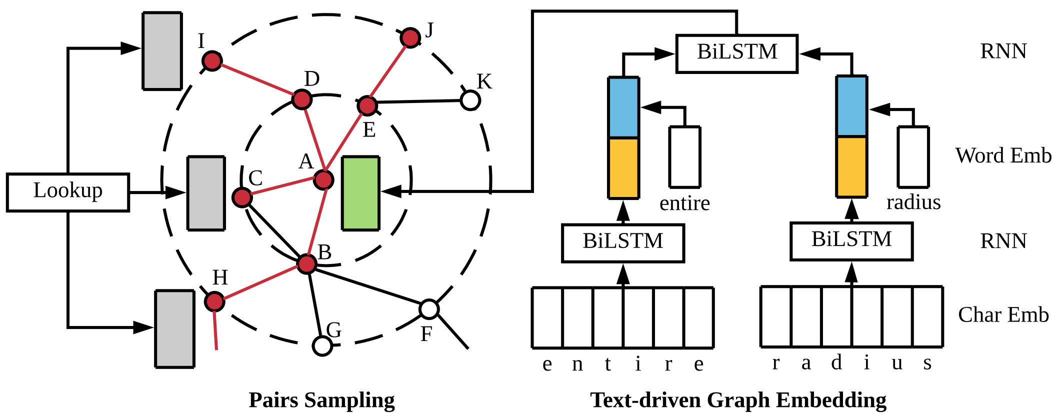

Figure 2: Model architecture of TGE-PS. *Note:* Circles represent nodes and colored boxes represent embeddings. Blue boxes represent word-level embeddings, yellow boxes represent character-based word embeddings, green boxes represent central embeddings and gray boxes represent neighbor embeddings. The sampled nodes and paths are colored as red.

where s denotes a node sequence, S is the set of all sequences, ω_i^s is the abbreviation of the window function $\omega(v_i, s)$ which denotes the nodes within the window of v_i when v_i appears in s and j' denotes the position of v_j in s, T denotes the sampling times of RW starting from v_i, T_i denotes times of v_i appearing in sequences except the one that v_i is the start node, $T_{j|i}$ denotes the amount of v_j appearing in N_i^k when v_i appears in sequence s and W denotes the window size. According to the objective, there are $T + T_i$ windows centering at v_i to be optimized in the sequences. Thus α_i is the ratio of v_i being more "important" than the other nodes, e.g., bridge nodes are likely to be sampled more frequently, and these nodes have larger α. $\beta_{\cdot|i}$ reflects the distribution of neighbor nodes appearing in training samples and $\beta_{j|i}$ is the probability of v_j sampled in $\mathcal{N}_i^o$. Induced from proofs in TADW, in DeepWalk $\beta_{*|i}$'s are actually the non-zero elements of the i-th row of transition matrix M in DeepWalk, or biased transition matrixM' in node2vec. Therefore, $\beta_{j|i}$ also reflects transition probability from v_i to neighbor node v_j.

Till now, we get the ideal general training objective of RW-based models regardless of the normalizing term. Now we analyze the sampling inefficiency of RW in two aspects:

Biased Objective. Comparing Eq. (1) and (2), we find node2vec is biased from its log-probability objective since each neighborhood N_i^k are weighted by different α_i, unless α_i is equal for each node. However, as RW adopts a sliding window to generate pairs from sampled sequences, the number of training pairs of v_i increases every time it is visited, and v_i will have a larger α_i if it is more frequently visited. Hence node2vec introduces a prior distribution implicitly, making its training objective biased from its proposal in Eq. (1). If we follow Eq. (1) strictly, α should be all zero and we should only sample each neighborhood T time, which can reduce a lot of training samples.

Interconnected Neighbors. $\beta_{j|i}$ converges to the biased transition probability from v_i to v_j with sufficiently large number of samples. Transition probability from v_i to v_j is mainly affected by interconnections between neighbor nodes, and in this case RW in fact introduces an assumption that during training embeddings of central nodes, interactions between neighbor nodes should be completely considered. Under this assumption, different neighbors have different probability to be sampled, and thus to sample neighbor nodes with small degree may result in many more samples of those with large degree. Therefore, the ideal sample complexity of a neighborhood $\mathcal{N}_i^k$ is determined by the minimal sampling probability $\beta_{j*|i}$, i.e., $O(1/\beta_{j*|i})$ where v_{j*} is least possible to be sampled , by assuming every node in $\mathcal{N}_i^k$ is sampled at least a minimum times to ensure sufficient training. This complexity must be greater than $O(|\mathcal{N}_i^k|)$ since $\beta_{j*|i} \leq 1/|\mathcal{N}_i^k|$. However, it is a free lunch to assume when optimizing a central node v_i, its neighbor nodes $v_j \in N_i^k$ have already been trained and their embeddings have already been encoded with the structural information respectively. This assumption must hold true after training RW for some time, otherwise the target of node embedding becomes ill-posed and can never be achieved. Thus central-neighbor connections are much more important than neighbor-neighbor connections, and neighbor-neighbor connections within one neighborhood in fact will be considered when they become the central-neighbor connections in other neighborhoods. Therefore, adopting a stronger assumption that alleviates variance of distribution $\beta_{\cdot|i}$ is still possible to produce good results while decreases sampling complexity.

Method Introduction The intuition of Pairs Sampling (PS) has two points: For the **biased objective** problem, we sample each neighborhood the same times and introduce higher-order Markov property to prevent a neighbor node from being re-sampled; For the **interconnected neighbors** problem, we sample central-neighbor pairs directly from a neighborhood and ignore neighbor-neighbor connections.

For every node v_i, we call one node v_j^o as an o-th order neighbor of v_i when the shortest distance between v_i and v_j is o. To obtain the training node pairs set $\mathcal{P}$, we sample the neighbor nodes in $\mathcal{N}_i^O$ in the following process:

1. Add all first-order neighbors of v_i into the node set $\mathcal{N}$.

2. If $O > 1$, for every v_j^o $(1 \leq o \leq O-1)$, sample a next-order neighbor node v_k^{o+1} from the following distribution:

$$p(v_k^{o+1}|v_j^o) = \begin{cases} \frac{1}{Z} & \text{if } (v_j^o, v_k^{o+1}) \in E \text{ and} \\ & dist(v_i, v_k^{o+1}) = o+1 \\ 0 & \text{otherwises.} \end{cases}$$

where Z is the number of $(o+1)$-order neighbors connected to v_j^o. The sampled v_k^{o+1} is added into $\mathcal{N}$.

3. For each node $v_j \in \mathcal{N}$, add node pair (v_i, v_j^o) into $\mathcal{P}$.

4. Repeat for N times

We take the graph in Figure 2 as an example to illustrate how PS works. In this graph, A is the central node, with dashed circles indicating the neighbors of the same order. Firstly, first-order neighbors $\{B, C, D, E\}$ are all sampled. H is sampled from $\{F, G, H\}$ as the successor of B. I is sampled as the successor of D. J is sampled from $\{J, K\}$ as the successor of E. Unsampled nodes are ignored in the next iteration of sampling, only sampled nodes in this order can be used to generate the next-order samples, e.g., H continues searching while F stops. By now PS samples a set of node pairs $\{(X, A)\}|X \in \{B, C, D, E, H, I, J\}\}$.

When adopting PS, the training objective becomes

$$\max_\theta N \sum_{v_i \in V} \sum_{o \in \{1,2,\ldots,O\}} \sum_{v_{j'} \in \mathcal{N}_i^o} \mathbf{e}_{j'}^\top \mathbf{e}_i$$

where $v_{j'}$ is the j'-th sampled node in $\mathcal{N}_i^o$. By restricting the max order of neighbors and the number of successors for each node to be at most 1, we successfully set an upper bound for the total number of node pairs. We also try other variants of PS like sampling the next-order under a constant probability instead of a constant number, or uniformly sampling neighbor nodes in the same order regardless of those in the former order. However, these policies are harder to control the sample complexity (since high-order neighbors increase exponentially) or ignore too much connections between neighbors.

Text-driven Graph Embedding method

Intuition Previous graph embedding models focus on generating graph embeddings from only structural information (denoted as NE) or incorporating text attributes with structural information (denoted as NE+TE). NE+TE models can be regarded as encoding structural and textual information into an NE space and a TE space respectively and generate final node representations from the interactions between these two spaces. Apart from NE or NE+TE, generating graph embeddings from text embeddings (denoted as TE2NE) is another practical method. We compare the three methods in Table 1, where $|\mathcal{V}|$, $|\mathcal{D}|$ and $|\bar{T}|$ represent the total number of nodes, the size of dictionary and the average length of texts respectively.

TE2NE is more suitable for large graphs with rich text information and strong text dependency. A large-scale graph may contain million- to billion-level nodes, where the number of nodes is much larger than the number of words. Besides, TE2NE can also apply to zero-shot scenarios, since no explicit NE is required in inference. Therefore, we propose the Text-driven Graph Embedding (TGE) method that makes the most of textual information by projecting textual information into the NE space. To model the text, we adopt bidirectional LSTM (BiLSTM) (Ma and Hovy 2016) for its success in Nature Language Processing field (Mikolov and Zweig 2012; Mikolov et al. 2010). To deal with out-of-vocabulary words in unseen nodes, we adopt character-level embeddings. Character-level embeddings have proven success in NLP tasks (Chung, Cho, and Bengio 2016; 2016; Ma and Hovy 2016). The advantages of Character-level embeddings are summarized by (Chung, Cho, and Bengio 2016) as: (i) good performance in out-of-vocabulary scenarios, (ii) ability to capture morphological features of language and (iii) no need for segmentation. Hence, we adopt character-level embeddings in addition to word-level embeddings.

Table 1: Comparison among Different Methods.

	NE	NE+TE	TE2NE
Space Complexity	$O(\|V\|)$	$O(\|V\| + \|\mathcal{D}\|)$	$O(\|\mathcal{D}\|)$
Time Complexity	$O(1)$	$O(\|\bar{T}\|)$	$O(\|\bar{T}\|)$
Zero-shot Scenarios	×	×	✓
Text Dependency	No	Weak	Strong

Generating Embeddings We denote the set of words as $\mathcal{D}^w$, the set of characters as $\mathcal{D}^c$ and text of node v_i to be $t_i = \{w_{ij}, 1 \leq j \leq |t_i|\}$, where $w_{ij} \in \mathcal{D}^w$ and $|t_i|$ is the length of t_i. Each word w_{ij} contains characters as $w_{ij} = \{c_{ijk}, 1 \leq k \leq |w_{ij}|\}$, where $c_{ijk} \in \mathcal{D}^c$ and $|w_{ij}|$ is the length of w_{ij}. We denote word- and character-level embedding vectors as $\mathbf{e}^w$ and $\mathbf{e}^c$ respectively.

We start by generating embedding of node v_i. We feed the sequence of character embeddings $\mathbf{e}^c_{ij,1:|w_{ij}|}$ into character-level BiLSTM and obtain a character-based word embedding $\mathbf{e}^{w'}_{ij}$ as

$$\begin{aligned} \overrightarrow{\mathbf{e}}^{w'}_{ij} &= \text{LSTM}^c_f(\mathbf{e}^c_{ij,1:|w_{ij}|}) \\ \overleftarrow{\mathbf{e}}^{w'}_{ij} &= \text{LSTM}^c_b(\mathbf{e}^c_{ij,|w_{ij}|:1}) \\ \mathbf{e}^{w'}_{ij} &= [\overrightarrow{\mathbf{e}}^{w'}_{ij}; \overleftarrow{\mathbf{e}}^{w'}_{ij}]. \end{aligned}$$

The character-based word embedding is concatenated with the corresponding word embedding $\mathbf{e}^w_{ij}$ and fed into the word-level BiLSTM layer as

$$\begin{aligned} \overrightarrow{\mathbf{e}}^h_i &= \text{LSTM}^w_f([\mathbf{e}^w_{i,1:|T_i|}; \mathbf{e}^{w'}_{i,1:|T_i|}]) \\ \overleftarrow{\mathbf{e}}^h_i &= \text{LSTM}^w_b([\mathbf{e}^w_{i,|T_i|:1}; \mathbf{e}^{w'}_{i,|T_i|:1}]) \\ \mathbf{e}_i &= \tanh(W[\overrightarrow{\mathbf{e}}^h_i; \overleftarrow{\mathbf{e}}^h_i] + b). \end{aligned}$$

Now we obtain the text-based node embedding $\mathbf{e}_i$ of node v_i. We also set up a lookup layer which embeds v_i into dim-dimension structure-based vector $\mathbf{e}'_i$ that is used to help train $\mathbf{e}_i$, and outputs $\mathbf{e}_i$ as the embedding vector of v_i.

Training To reduce computation complexity, we adopt Negative Sampling (Mikolov et al. 2013) like node2vec does and define the loss function to be

$$\mathcal{L}_{\text{sim}}(v_i, v_j) = -\log(\sigma(\mathbf{e}'^\top_j \mathbf{e}_i)) - \sum_{v_k \sim P(v)}^{N_{\text{neg}}} \log(\sigma(-\mathbf{e}'^\top_k \mathbf{e}_i)),$$

where $\mathbf{e}_i$, $\mathbf{e}'_j$ and $\mathbf{e}'_k$ denotes embeddings of the central node, the neighbor node in a pair, and the randomly sampled negative node, N_{neg} denotes the number of negative samples, σ denotes the sigmoid function and $P(v) \propto d_v^{3/4}$ denotes the distribution of nodes when sampling negative samples, where d_v is the degree of v.

We also apply L_2-regularization on parameters and embeddings

$$\mathcal{L}_{\text{reg}}(v_i, v_j) = \sum \|w\|_2^2 + \|\mathbf{e}_i\|_2^2 + \|\mathbf{e}'_j\|_2^2 + \sum_k^{N_{\text{neg}}} \|\mathbf{e}'_k\|_2^2. \quad (3)$$

The final objective is to minimize

$$\mathcal{L} = \sum_{(v_i,v_j)\in\mathcal{P}} (\mathcal{L}_{\text{sim}}(v_i, v_j) + \lambda\mathcal{L}_{\text{reg}}(v_i, v_j)), \quad (4)$$

where λ is a hyper-parameter. We employ AdaGrad (Duchi, Hazan, and Singer 2011) to minimize the loss.

Experiments

Datasets

To verify the effectiveness and efficiency of PS, we conduct a series of experiments including link prediction and node classification over the following 7 datasets. We evaluate TGE-PS on link prediction tasks over 2 datasets. These datasets include **Cora** (McCallum et al. 2000), **Facebook** (Leskovec and Krevl 2014), **Zhihu** (Tu, Liu, and Liu 2017), **BlogCatalog (BlogCat)** (Zafarani and Liu 2009), **arXiv AstroPh (AstroPh)** (Leskovec and Krevl 2014), **arXiv HepTh (HepTh)** (Leskovec and Krevl 2014) and **Systemized Nomenclature of Medicine - Clinical Terms (SNOMED CT)** (Donnelly 2006). We list their details in Table 2 where $|V|$ and $|E|$ refer to number of nodes and edges, respectively. Apart from details in the table, Cora and BlogCat have 7 and 39 types of labels respectively. We use the abstract of each paper as the textual information in HepTh and the longest description as textual information of each node in SNOMED. Before conducting our experiments, the preprocessing includes lowering all characters, removing stop words and discarding punctuations. In practice, We keep all nodes and $p\%$ edges of the dataset $Data$ for training (denoted as $Data@p\%$).

Table 2: Details of Datasets

Datasets	$\|V\|$	$\|E\|$	Type
Cora	2,211	5,214	Citation Graph
Facebook	4,039	88,234	Social Network
Zhihu	10,000	43,894	Q&A Datasets
BlogCat	10,312	333,983	Blog Directory
AstroPh	18,772	198,110	Co-work Network
HepTh	27,400	352,542	Citation Graph
SNOMED	391,892	2,047,749	Health Terminology

Baselines

We evaluate our model against several graph embedding models : **LINE** (Tang et al. 2015), **DeepWalk** (Perozzi, Al-Rfou, and Skiena 2014), **node2vec** (Grover and Leskovec 2016), **TADW** (Yang et al. 2015), **CANE** (Tu, Liu, and Liu 2017), **Paper2Vec** (Ganguly and Pudi 2017) and **STNE** (Liu et al. 2018). We also design a rule-based baseline embedding model:

Text Matching represents each node as the average vector of pre-trained word embeddings from Glove (Pennington, Socher, and Manning 2014) for every word in the text. The performance of Text Matching reflects the dependency between graph and text structure.

Evaluation Metrics

For link prediction, we adopt AUC (Area Under Curve) (Hanley and McNeil 1982) to evaluate the embeddings and compute the scores in two different ways:

- We use the same number of positive and negative pairs to train a logistic regression classifier. For each pair, the classifier inputs the Hadamard products of the embeddings, and outputs the binary classification results. We use the trained classifier to infer the connectivity of pairs in another set and compute the final AUC score, which we denote as AUC$_{\text{LR}}$.
- We compute the score of each node pair in test set by

$$score(e_c) = \begin{cases} 1/N_{test} & \text{if } \mathbf{e}_c^\top \mathbf{e}_p > \mathbf{e}_c^\top \mathbf{e}_n \\ 0.5/N_{test} & \text{if } \mathbf{e}_c^\top \mathbf{e}_p = \mathbf{e}_c^\top \mathbf{e}_n \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

  where N_{test} is the number of tested nodes. By adding up scores of all tested nodes we obtain the final AUC score, which we denote as AUC$_{\text{pair}}$.

AUC$_{\text{LR}}$ performs better in capturing potential relation between embeddings but may suffer from overfitting problem, while AUC$_{\text{pair}}$ is a naive yet solid measurement of embedding results. Thus, we report the overall best results in two columns, which represent AUC$_{\text{LR}}$ and AUC$_{\text{pair}}$, respectively.

For node classification, we report the macro-F_1 scores over all classes as previous works do.

Experiments of Pairs Sampling

Theoretical Analysis Firstly, we compute the number of sampled pairs of each method. Given the number of nodes $|V|$, average degree $\bar{d}$, walk length L, window size W and walk time T in RW, and max order O and sampling time N in PS, we present the expressions of node pairs number in Table 3.

Table 3: Expressions of Node Pairs Number

	Random Walk	Pairs Sampling
# pairs	$(2L - W - 1)WT\|V\|$	$NO\bar{d}\|V\|$

We compute the number of sample pairs under RW and PS and use r to represent the ratio of them as $\frac{(2L-W-1)WT}{NO\bar{d}}$. Our comparison is conducted over several preprocessed datasets. Specially, we keep only 20% edges of SNOMED for these reasons: (i) even under this setting there are over 400,000 edges in the training data; (ii) experiment results show that 20% edges are sufficient for producing good results. For the fair comparison of the efficiency, we apply grid search on L, W, T, O and N and generate corresponding training sets for RW and PS. We select parameters with the best performance on each dataset and compute the ratio of node pairs. The ratios are reported in Table 4. Note that the real ratios will be larger for that the number of pairs sampled by PS is actually equal to or smaller than $NOd|V|$.

Table 4: Ratios of Different Datasets

Datasets	$\lvert V\rvert$	$\lvert E\rvert$	d	r
Cora@50%	2,211	2,607	2.33	183.60
Cora@100%	2,211	5,214	4.42	140.46
Facebook@50%	4,039	44,117	21.84	36.17
Zhihu@50%	10,000	21,947	4.28	369.16
BlogCat@100%	10,312	333,983	64.78	57.50
AstroPh@50%	18,772	99,054	10.56	55.40
HepTh@50%	27,400	176,271	12.86	99.79
SNOMED@20%	391,892	409,550	2.09	409.36

From Table 4 we can see that $(2L - W - 1)WT$ is much larger than NOd and PS can significantly reduce the training samples (reducing ~99% samples) compared with RW. Note that r is often much larger in sparse networks like Cora, Zhihu and SNOMED. In graphs with small average degree, RW often conducts DFS-like walks and encounter leaf nodes, which results in frequent revisiting behaviors and hence more redundant samples.

Link Prediction In traditional link prediction settings, a portion of edges are removed in training set while ensuring each node connected with at least one edge. The removed edges are used as test set. We evaluate PS against RW on various datasets. We use PS and RW to generate training sets, and train node embeddings under the same framework. The results are shown in Table 5.

Table 5: Link Prediction Results of RW and PS

Datasets	Random Walk		Pairs Sampling	
	AUC_{LR}	AUC_{pair}	AUC_{LR}	AUC_{pair}
Cora@50%	0.9200	0.9293	**0.9272**	**0.9394**
Facebook@50%	0.9921	0.9892	**0.9922**	**0.9911**
Zhihu@50%	0.8659	**0.9144**	**0.8673**	0.9136
AstroPh@50%	0.9788	0.9768	**0.9795**	**0.9789**
HepTh@50%	0.9741	0.9648	**0.9743**	**0.9730**
SNOMED@20%	0.9350	0.9396	**0.9359**	**0.9402**

Results in Table 5 show that:

- PS outperforms RW on almost all datasets. Experimental results indicate that even if RW consider extra information like interconnection between neighbors and PS adopts stronger assumptions for efficiency consideration, PS still proves to be a competitive alternative to RW.
- In some datasets like Facebook and Zhihu where $W = 1$ and $O = 1$, PS and RW is quite similar in performance. However, PS generates fewer number of pairs and still presents advantages in sampling efficiency.

Node Classification We evaluate PS against RW on one small graph, Cora and another large graph, BlogCat. We use all the nodes and edges to train node embeddings, and randomly select 50% node labels to train the multi-label multi-class SVM classifier, leaving the other 50% labels for testing. Other settings are the same as the previous section. We report the results in macro F_1-Score in Table 6. Results in Table 6 show that PS is at least competitive against RW on node classification task.

Table 6: Node Classification Results of RW and PS

Dataset	Random Walk	Pairs Sampling
Cora@100%	0.8079	**0.8085**
BlogCat@100%	**0.2581**	0.2544

Parameter Sensitivity Max order O and sampling time N are the major parameters of PS, and we conduct experiments to analyze their parameter sensitivity. We study the performance of PS on link prediction over Facebook, AstroPh and HepTh. For each dataset, we fix N at first to evaluate $O = 1, \dots, 8$, and then we use the best O to evaluate $N = 1, \dots, 10$. We plot the results in Figure 3 and Figure 4.

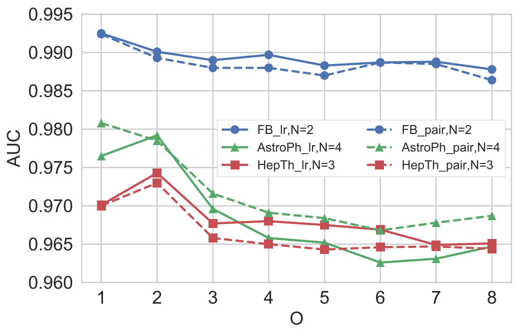

Figure 3: Parameter sensitivity (max order O).

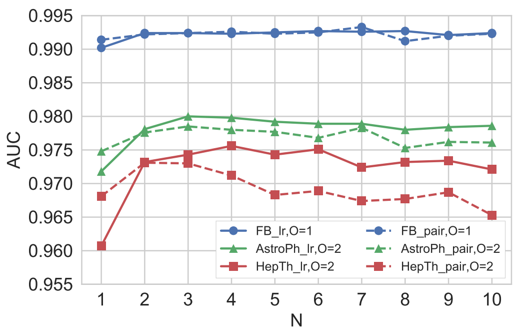

Figure 4: Parameter sensitivity (sample time N).

From Figure 3 and 4 we can see that:

- Figure 3 shows that graph embedding models perform better with a relatively small O. Larger value of O introduces harmful noise and is harmful especially in small graphs.
- It is interesting that AUC scores of Facebook remain invariant against N in Figure 4. This is reasonable for that O of Facebook is set to be 1, under which setting increasing N adds only duplicates node pairs and has little impact on AUC scores.

Experiments of TGE-PS

Link Prediction We evaluate TGE-PS on SNOMED and HepTh respectively. The results are shown in Table 7. We classify baseline models by with or without text utilization and present them in two large groups.

Table 7: Link Prediction Results of Different Models

Model	SNOMED@20%		HepTh@50%	
	AUC_{LR}	AUC_{pair}	AUC_{LR}	AUC_{pair}
LINE	0.6461	0.6985	0.7883	0.7520
DeepWalk	0.9164	0.9258	0.9711	0.9573
node2vec	0.9350	0.9396	0.9741	0.9648
Text Matching	0.8954	0.8717	0.7057	0.5700
TADW	-	-	0.8866	0.8977
CANE	0.9613	0.9544	0.9785	0.9388
Paper2Vec	0.9581	0.9604	0.9745	0.9748
STNE	-	-	0.9651	0.9572
PS	0.9359	0.9402	0.9758	0.9716
TGE-PS	**0.9721**	**0.9621**	**0.9793**	**0.9752**

In Table 7, "-" refers to results of failed experiments due to OOM. From Table 7, we have following observations:

- TGE-PS outperforms all baseline models on both datasets. This result indicates that generating node embeddings from text embeddings is practical in real-world datasets.
- Incorporating textual information with structural information helps constructing expressive graph embeddings. Improvements of TGE-PS over PS or Paper2Vec over node2vec strongly support this point. Only preserving structural information like LINE or textual information like Text Matching both present limitations.
- The improvements of TGE-PS are quite different on the two graphs. Recall that we use the text description of concepts in SNOMED, and we use the whole abstract in HepTh, we owe this to that SNOMED has stronger text dependency than HepTh. The performance of Text Matching in SNOMED supports this point. Similar difference of improvements also appears between Paper2Vec and node2vec.

Zero-Shot Link Prediction Unlike common link prediction scenarios, where each node embedding is trained at least once, zero-shot scenarios require inference on link existence with unseen nodes. Zero-shot scenarios are common and important in real-world applications. Only TE-only and TE2NE can be applied in these scenarios while NE-only and NE+TE can not. However, to better study this problem, we manage to conduct experiments on CANE by training with whole model and evaluating it with TE part only, which we denote as CANE (TE).

We conduct zero-shot link prediction experiments on SNOMED and HepTh where 0.5% of the nodes and related edges are removed from the graph. We also conduct ablation experiments on TGE-PS to analyze the influence of character- and word-level embeddings. In ablation experiments, we either: (i) remove word-level embeddings and replace $[\mathbf{e}^w; \mathbf{e}^{w'}]$ with $\mathbf{e}^{w'}$, or (ii) remove character-level embeddings and replace $[\mathbf{e}^w; \mathbf{e}^{w'}]$ with $\mathbf{e}^w$. We use '-w' and '-c' to denote the two settings respectively. The results are shown in Table 8.

Table 8: Zero-Shot Link Prediction Results

Model	SNOMED		HepTh	
	AUC_{LR}	AUC_{pair}	AUC_{LR}	AUC_{pair}
Text Matching	0.9059	0.8813	0.5934	0.6250
CANE (TE)	0.5271	0.5344	0.6036	0.5288
TGE-PS (-w)	0.5000	0.5003	0.5000	0.5037
TGE-PS (-c)	0.9701	0.9786	0.8979	**0.9485**
TGE-PS	**0.9760**	**0.9811**	**0.8990**	**0.9485**

From Table 8, we have the following observations:

- The high AUC scores indicate TGE-PS is practical and reliable in zero-shot scenarios. The unstable performance of CANE (TE) indicates that without trained NE, the model is incapable of making reasonable inference from textual information and may perform even worse than Text Matching.
- Word-level embeddings are essential in embedding nodes. This is not surprising given that there are much more words (100,471 in SNOMED and 72,083 in HepTh) than characters (88 in SNOMED and 59 in HepTh).
- The improvements of character-level embeddings are higher in SNOMED. We believe this is caused by characteristic of medical terms in SNOMED that are often composed of sub-words. As we stated before, character-level embedding excels in capturing such information.

It is also worth noting that graph embedding models differ in the capacity to deal with large graphs. All models in Table 7 can handle HepTh, but some fail on SNOMED. We also try to conduct experiments on GraphGAN in link prediction task, but they fail on both datasets. Besides, we conduct experiments training TGE with node pairs sampled by RW and they fail for taking extremely long time to finish training on both datasets. Our TGE-PS performs well in handling large graphs like SNOMED@20%, and can even be trained in zero-shot scenario as presented.

Conclusions and Future Works

In this paper, we propose a novel Text-driven Graph Embedding with Pairs Sampling (TGE-PS) framework. TGE-PS includes two stages: sampling and embedding. In sampling stage, we propose Pairs Sampling (PS) strategy countering drawbacks of popular Random Walk. PS reduces $\sim$99% samples with competitive results on 7 datasets. In embedding stage, we propose Text-driven Graph Embedding (TGE) method that generates embeddings from text embeddings while preserving the structural information. TGE-PS surpasses previous graph embedding models on link prediction task, and produces remarkable results in zero-shot scenarios.

We will explore the following research directions in future: (I) We will study on sampling strategy that converges to more valid distributions of α_i and $\beta_{j|i}$. (ii) We will improve performance of TGE-PS on graphs with long texts like the whole body of articles.

References

Ahmed, A.; Shervashidze, N.; Narayanamurthy, S.; Josifovski, V.; and Smola, A. J. 2013. Distributed large-scale natural graph factorization. In *22nd WWW*, 37–48. ACM.

Cai, H.; Zheng, V. W.; and Chang, K. 2018. A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE Transactions on Knowledge and Data Engineering*.

Cao, S.; Lu, W.; and Xu, Q. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 891–900. ACM.

Cao, S.; Lu, W.; and Xu, Q. 2016. Deep neural networks for learning graph representations. In *AAAI*, 1145–1152.

Chung, J.; Cho, K.; and Bengio, Y. 2016. A character-level decoder without explicit segmentation for neural machine translation. *arXiv preprint arXiv:1603.06147*.

Donnelly, K. 2006. Snomed-ct: The advanced terminology and coding system for ehealth. *Studies in health technology and informatics* 121:279.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR* 12(Jul):2121–2159.

Ganguly, S., and Pudi, V. 2017. Paper2vec: Combining graph and text information for scientific paper representation. In *ECIR*, 383–395. Springer.

Goyal, P., and Ferrara, E. 2017. Graph embedding techniques, applications, and performance: A survey. *arXiv preprint arXiv:1705.02801*.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable Feature Learning for Networks.

Hanley, J. A., and McNeil, B. J. 1982. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology* 143(1):29–36.

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Leskovec, J., and Krevl, A. 2014. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

Liu, J.; He, Z.; Wei, L.; and Huang, Y. 2018. Content to node: Self-translation network embedding. In *24th SIGKDD*, 1794–1802. ACM.

Ma, X., and Hovy, E. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

McCallum, A. K.; Nigam, K.; Rennie, J.; and Seymore, K. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3(2):127–163.

Mikolov, T., and Zweig, G. 2012. Context dependent recurrent neural network language model. *SLT* 12:234–239.

Mikolov, T.; Karafiát, M.; Burget, L.; Černockỳ, J.; and Khudanpur, S. 2010. Recurrent neural network based language model. In *11th ISCA*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Nishana, S., and Surendran, S. 2013. Graph embedding and dimensionality reduction-a survey. *IJCSET* 4(1):29–34.

Ou, M.; Cui, P.; Pei, J.; Zhang, Z.; and Zhu, W. 2016. Asymmetric transitivity preserving graph embedding. In *22nd SIGKDD*, 1105–1114. ACM.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *EMNLP*, 1532–1543.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. DeepWalk: Online Learning of Social Representations.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. LINE: Large-scale Information Network Embedding.

Tu, C.; Liu, H.; and Liu, Z. 2017. CANE : Context-Aware Network Embedding for Relation Modeling. 1722–1731.

Wang, H.; Wang, J.; Wang, J.; Zhao, M.; Zhang, W.; Zhang, F.; Xie, X.; and Guo, M. 2017. GraphGAN: Graph Representation Learning with Generative Adversarial Nets.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 1225–1234. ACM.

Yang, C.; Liu, Z.; Zhao, D.; Sun, M.; and Chang, E. Y. 2015. Network Representation Learning with Rich Text Information. (Ijcai):2111–2117.

Zafarani, R., and Liu, H. 2009. Social computing data repository at ASU.