Noname manuscript No.
(will be inserted by the editor)

ShortWalk: An Approach to Network Embedding on Directed Graphs

Fen Zhao · Yi Zhang · Jianguo Lu

Abstract In network embedding algorithms, long random walks are often used to convert the graph into 'text' so that node embeddings can be learned by Skip-gram with Negative Sampling (SGNS) model. However, in a directed graph, long random walks can be trapped or interrupted, leading to low-quality embeddings. This paper proposes a new algorithm, called ShortWalk, to improve the directed graph network embeddings. ShortWalk performs short random walks that restart more frequently thus produces shorter traces. It also gives nodes equal weights by generating the training pairs using pair-wise combination of nodes on the traces. We validate our method on eight directed graphs with different sizes and structures. Experimental results confirm that ShortWalk outperforms DeepWalk consistently on all datasets in node classification and link prediction tasks.

Keywords Network embedding, Skip-gram, Negative Sampling, DeepWalk, Random Walk.

1 Introduction

Network embedding is crucial for network mining and analyses (Cai et al., 2018). Neural Network based network embedding algorithms, e.g., DeepWalk(Perozzi et al., 2014) and *node2vec* (Grover and Leskovec, 2016a), are based on the well-known word-embedding algorithm (Mikolov et al., 2013a). Among many variants of the cluster of the algorithms, the Skip-Gram with Negative Sampling model (SGNS) has been shown to be the most effective. It was originally designed for language modelling. Hence, a crucial step in a network embedding algorithm is transforming the network to a 'text' by a graph traversing method. DeepWalk uses long random walks. *node2vec* improved DeepWalk with biased random walks. Long random walks in DeepWalk and *node2vec* are necessary to produce 'text' so that subsequent SGNS can be applied. In undirected graphs, it is reported that the best performance is achieved when random walk length reaches 100 in most relevant papers, such as in (Grover and Leskovec, 2016a) and (Dong et al., 2017a).

School of Computer Science, University of Windsor, Windsor, Canada. E-mail: {zhao15m, zhang18f, jlu}@uwindsor.ca

However, random walks are normally not that long. The well-known PageRank algorithm uses random walks that restart at a probability ranging between 0.15 to 0.2 (Brin and Page, 1998; Zhao et al., 2019), resulting in an average walk length of five to six. Note that there are two crucial differences between random walk in PageRank and the random walk in DeepWalk: in PageRank, every path can be of different length, not a fixed length as in DeepWalk and *node2vec*; and the average path length is much shorter in PageRank.

There is a reason for choosing a shorter length in PageRank: a long random walk may be trapped in a small region, and some nodes could be visited repeatedly in the trapped area (Sen and Chaudhary, 2017). To avoid these traps, PageRank introduces damping factor that allows the walker teleports to another node randomly. There is also a reason for long random walks in DeepWalk and *node2vec*: long random walks resemble paragraphs in text, hence SGNS can be run on the 'text'. If paragraph lengths were only five on average, running SGNS would be meaningless since the window size is usually set from 5 to 10 (Perozzi et al., 2014).

To solve the dilemmas in directed graph embedding algorithms, we propose a new method called ShortWalk. It performs short random walks that have frequent restarts, resulting in short random walk traces. Then, instead of applying SGNS directly on the traces, ShortWalk obtains the training pairs with the pair-wise combination of the nodes in the short random walk traces. We prove that such pair-wise combination is actually equivalent to the sampling strategy in SGNS. We validate our method on eight directed graphs. Experimental results suggest ShortWalk outperforms DeepWalk consistently on all datasets in both classification and link prediction tasks.

Many graph embedding algorithms are based on SGNS, and only differ in the types of networks (e.g., signed networks, heterogeneous networks), and in the way to perform random walks (e.g., in favour of transitive relations, or proportional to Katz similarities). This paper focuses on the most simple network and the default random walk, to gain insight into the impact of short walk. We believe that our result can be extended to various types of networks and types of random walks.

2 Background and Related Work

Graph embedding or graph node embedding is to find a short and dense vector representation for nodes in a graph. We refer to (Wang et al., 2020) for a good review of recent developments in this area. Graph embedding is widely used in social network analyses, because once the embedding is obtained, off-the-shelve machine learning algorithms can be applied on graphs.

The basic idea of graph embedding is to find vector representation of the nodes so that their node similarities are preserved. The most straightforward approach is to use graph adjacency matrix to capture node similarities, and use matrix factorization, such as SVD (Singular Value Decomposition) and MDS(Multidimensional scaling), to reduce the dimensionality (Tenenbaum et al., 2000). Along this line, both the adjacency matrix and the dimensionality reduction techniques can be expanded. For the input matrix, instead of the zero/one adjacency matrix that simply reflects the edge relations in a graph, numerous other relations have been experimented with. For example, the input matrix can be Laplacian matrix (Belkin and Niyogi, 2002), Katz matrix that reflects Katz similarities (Ou et al., 2016),

multiple-hop connections (Cao et al., 2015). For the dimensionality reduction, SVD has the scalability issue. More approaches have been treating dimensionality reduction as an optimization problem that can be solved with SGD algorithms (Goyal and Ferrara, 2018a) (Cai et al., 2018).

Graph embedding induced from SGNS With the breakthrough in neural-network algorithms (Bengio et al., 2003), in particular the negative sampling techniques inspired by noice contrastive estimation (Gutmann and Hyvärinen, 2010), the state-of-the-art of graph embedding algorithms are derived from SGNS(Skip-Gram Negative Sampling). SGNS was first proposed as a variant of word2vec algorithms (Mikolov et al., 2013b). The representative graph embedding algorithm in this direction is DeepWalk (Perozzi et al., 2014). It produces random walk traces to feed into SGNS. Node2Vec argues that the default random walk may not be able to capture the structure of the graph, and proposes a range of biased random walks that are controlled by hyper-parameters to be adjusted according to graph structure (Grover and Leskovec, 2016b). LINE produces the training pairs for SGNS similar to Katz distances that are mainly of first-order and second order proximity (Tang et al., 2015a). Although these algorithms claim to work on directed graphs as well, but the results are not good as we will demonstrate in the experiment section.

Directed graph embedding There has been substantial work on embeddings designed specifically for directed graphs. Some are performed in the setting of matrix factorization, such as HOPE (Ou et al., 2016), ATP (Sun et al., 2018), and DGE (Chen et al., 2007). They focus on the comparison of various input similarity matrices. For instance, HOPE compared extensively on many node similarity metrics including Katz Index, Rooted Page Rank, Common Neighbors, and Adamic-Adar score. Then matrix factorization is applied to obtain node embeddings. ATP (Sun et al., 2018) preserves the asymmetric relations by incorporating the hierarchy matrix and adjacency matrix together. NMF(Non-negative matrix factorization) is used to factorize the asymmetric matrix into embeddings.

Recently, directed graph embedding algorithms that are induced from SGNS start to emerge, e.g., APP(Zhou et al., 2017) and NERD(Khosla et al., 2019). They try to embed graph nodes to two spaces, one for target and one for source, capturing the directionality of the edges. This could be useful in some applications such as link prediction where relations are not symmetric. Our ShortWalk seeks to represent nodes in one space so that it is more generic. APP uses the rooted random walk to generate a path from a source to a target to preserve the asymmetric proximity. The $(source, target)$ pairs are fed into the SkipGram with Negative Sampling(SGNS) model to learn network embeddings. Similarly, NERD proposes an alternating random walk to generate training pairs by defining a source walk and target walk. There are also embeddings for more complex networks, such as signed and directed network (Kim et al., 2018), and heterogeneous information networks (Dong et al., 2017b). (Abu-El-Haija et al., 2017) experimented on directed graphs but mainly on edge embedding. VERSE (Tsitsulin et al., 2018) also explores PageRank like random walks among other features.

Walk length Most existing works use random walk with fixed length to capture the structure of a network. The walker does not stop on current path unless the length of the path reaches a threshold l. It is widely used by network embedding algorithms such as DeepWalk (Perozzi et al., 2014), *node2vec* (Grover and Leskovec, 2016a), walklets (Perozzi et al., 2016), and metapath2vec (Dong et al.,

2017a), etc.. Existing works also suggest that longer path will give better results in the undirected graphs (Grover and Leskovec, 2016a; Dong et al., 2017a). Thus, the length of the path l is typically set to a large number such as 40 (Perozzi et al., 2014), 80 (Grover and Leskovec, 2016a), or 100 (Dong et al., 2017a). Intuitively, the random walk with fixed length is a variant of standard random walk with $\alpha = 1 - 1/l$. Taking DeepWalk as an example. When $l = 100$, DeepWalk equals to random walk with damping factor $\alpha = 1 - 1/100 = 0.99$. However, in most PageRank applications, such as search engine for World Wide Web, the damping factor α is set to 0.85 (Brin and Page, 1998), which is much smaller than DeepWalk. The difference may not be obvious in an undirected graph, where the PageRank values are proportional to nodes' degrees (Grolmusz, 2015). However, in directed graphs, the larger α will cause problems on certain graph structures such as spider traps (Sen and Chaudhary, 2017). The details will be discussed in the next section.

3 The Dilemma

3.1 The Problem of Long Random Walks

Long random walk is never an option in PageRank-like algorithms (Brin and Page, 1998; Zhao et al., 2019). The reason is obvious: enclosed loops may occur and nodes could be visited repeatedly in long random walks, hence the visiting probability can be enhanced tremendously. For instance, if two webpages link each other only, each would be visited 50 times if the walk length is 100 that is the default parameter in typical network embedding algorithms like DeepWalk (Perozzi et al., 2014). The 'importance' of the nodes are amplified by roughly 50 times, compared with the nodes that connect well with others. To overcome this problem, PageRank-like algorithms introduce frequent random jumps, say, random jump with a probability of 0.15, resulting in average random walk length of around six. Thus, the importance of the mutually linked webpages would be amplified by 6/2=3 on average, instead of 50 in long random walks in this example.

Table 1 Top visited nodes in WebGoogle. Nodes (webpages) are sorted by their occurrence count in DeepWalk^{100} traces.

Node	Degree		DeepWalk^{100}		DeepWalk^{5}	
	In	Out	Count (/total)	PageRank ($\alpha = 0.99$)	Count (/total)	PageRank ($\alpha = 0.8$)
www.google.com/googleblog/	203	1	0.103	0.101	0.002	0.004
www.google.com/advanced_search	11,397	11	0.065	0.054	0.082	0.064
www.google.com/support/talk	5	1	0.056	0.076	0.001	0.002
www.google.com/holidaylogos.html	7,730	15	0.047	0.037	0.046	0.037
www.google.com/terms_of_service.html	3,384	10	0.020	0.028	0.015	0.020
www.google.com/intl/en/about.html	270	9	0.016	0.011	0.012	0.009
www.google.com/intl/en/ads/	98	15	0.015	0.012	0.012	0.010
www.google.com/intl/en/services/	49	17	0.015	0.011	0.011	0.009
www.google.com/webmasters/	1,036	21	0.014	0.016	0.010	0.011
www.google.com/options/	3,679	5	0.013	0.013	0.017	0.016

Long random walks indeed cause such unfair weighting in real applications, in various kinds of graphs. We illustrate this phenomenon with a real webpage datasets crawled from *http://google.com* (Palla et al., 2007). Table 1 lists the occurrences for top 10 visited webpages. The superscript of DeepWalk represents the maximum length of the traces. Webpages are sorted by their occurrence in DeepWalk with walking path $l = 100$. DeepWalk uses random walk with fixed length to generate the traces. It can be treated as a variant of PageRank with $\alpha = 1 - 1/l$. Therefore, we also list the corresponding PageRank value in the table. From the table, we can see that *Google Blog* occupies 10.3% occurrence in the walk traces. Therefore, it will have massive training pairs in SGNS. However, this webpage has a very small in-degree and out-degree – 203 and 1. We notice that this node has a very high PageRank value calculated by $\alpha = 0.99$, which is very close to its occurrence. Moreover, the Pearson's Correlation Coefficient between a node appears in the walking traces generated by DeepWalk^{100} and its corresponding PageRank value by $\alpha = 0.99$ is 0.97. The high PageRank value is caused by the self-loop – this page has only one out-link pointing to itself. When setting $\alpha = 0.8$, the PageRank value of *Google Blog* becomes 25 times smaller than $\alpha = 0.99$. Thus, it is expected to see the occurrence obtained by DeepWalk^{5} decreases to 0.2%, which is 51.5 times less than long traces.

We then further explore the impact of path length l on PageRank value distribution using Figure 1. As shown in Panel (A), when l is large, the slope is steeper, gap between top occurred nodes and lower frequent nodes are big. It means that lower frequent node will have less training pairs than frequent ones. In this case, some nodes may not have enough training pairs, resulting in low-quality embeddings. On the other hand, the gap is much smaller when $l = 5$.

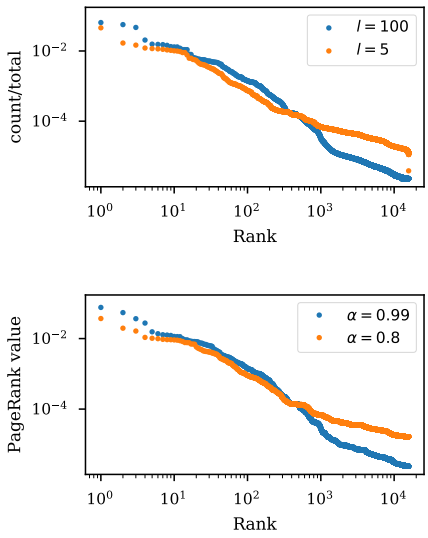


Fig. 1 Distributions of occurrence counts (Panel A) PageRank values (Panel B).

3.2 The Problem of Running SGNS on Short Random Walks

Since short random walks are more robust and reflect the node importance better, we may be tempted to run SGNS directly on traces of short random walks. The result is not satisfactory as demonstrated in the Section 5. This is caused by the strategy in SGNS for generating the training pairs as we explain below.

Given a typical short trace of length six $n_0n_1n_2n_3n_4n_5$. Suppose that five epochs are run in SGNS, and the window length is five as in a typical SGNS setting. When the center word is n_0, it will be used as the input of the neural network in SGNS and pairs with n_1, n_2, ... and n_5 with different frequencies, on average they are $5, 4, 3, 2, 1$ times respectively. When the center word moves forward, n_0 will be paired as output in the neural network with n_1 5 times, with n_2 4 times, and so on. Hence, altogether, n_0 occurs $5+4+3+2+1 = 15$ times as the input/output on average. On the other hand, nodes in the middle of the trace (or text in word2vec) will have higher occurrences. For instance, n_2 will be used as input in the neural network $4+5+5+4+3 = 21$ times as illustrated below:

$$\begin{cases} 4, & \text{when center word is } n_0 \\ 5, & \text{when center word is } n_1 \\ 5, & \text{when center word is } n_3 \\ 4, & \text{when center word is } n_4 \\ 3, & \text{when center word is } n_5, \end{cases} \tag{1}$$

It is much higher than that of n_0 (15). As a result, n_2 will be updated 1.4 times more than n_0 on average during the training process.

Those nodes should have equal importance – if they were in a long long trace, they would have the same occurrences in training. In short walks, the long trace was chopped down to shorter pieces for better visiting probability, but we should not penalize the nodes at the ends of a trace.

To summarize, node occurrence count in training pairs depends on two factors, one is the node visiting probability in random walks, the other is the scanning/sampling algorithm. When we change to short random walks to cater for more reasonable visiting probability, we also need to change the pair sampling algorithm to cope with the short trace.

4 Our method

4.1 Pair-wise Combination

To understand our solution, let us fall back to the word embedding problem in SGNS momentarily. Let us suppose that the text is continuous without paragraph or document breaks to simplify the discussion. Our problem is reduced to the following word embedding problem: If we scramble the text into short pieces, say each of length 5. With the long trace gone, what is the method to generate the training pairs from the scrambled pieces (5-grams)?

In this case, running SG on k-grams is not the right choice as we explained in the previous section. Instead, we should use co-occurrences Lund and Burgess (1996) that was used to capture word relations. Interestingly, the co-occurrence

count in k-grams is actually proportional to the SG count on long text. More specifically,

Theorem 1 *Given a long text and a pair of words w_i, w_j. Let $f_c(w_i, w_j)$ denote the co-occurrence frequency of (w_i, w_j) obtained from k-grams of the text, and $f_s(w_i, w_j)$ is the frequency obtained by Skip-Gram. $f_c(w_i, w_j)$ is proportional to the expectation of the $f_s(w_i, w_j)$, i.e.,*

$$f^c(w_i, w_j) \propto \mathbb{E}(f^s(w_i, w_j)) \tag{2}$$

Proof Suppose that w_i and w_j co-occur in a k-gram with x positions apart from each other, for $x < k-1$. They will co-occur in other neighbouring $k-x$ k-grams. In SG, when w_i is the centre word, (w_i, w_j) will be trained with probability $(k-x)/k$. Hence, the expected number of pairs in SG is proportional to $(k-x)$, supposing that k is a constant.

To support Theorem 1, we run SGNS and pair-wise combination on text8. Text8 is widely used to demonstrate word embedding algorithms, e.g., in (Pennington et al., 2014). It is the first 10^8 bytes of a clean dump from English Wikipedia. We set the window size to 5 in SGNS and sum the word pair occurrence of five runs. Then we compare the occurrence with the ones obtained by pair-wise combination in Figure 2. The x-axis is the rank of the training pairs and the y-axis is the corresponding occurrence. We can see that these two lines are matched perfectly with Pearson's Correlation Coefficient of 0.99. For instance, the most frequent word pair is (of, the). It appears 2,082,562 times in SGNS and 2,082,590 times in pair-wise combination.

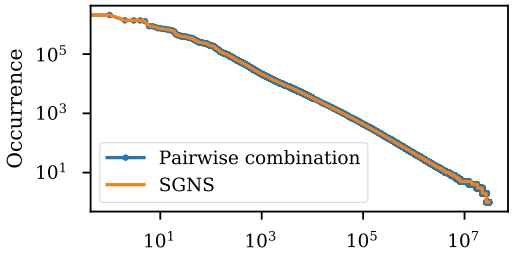


Fig. 2 Comparison of SGNS and pair-wise combination. The Pearson's Correlation Coefficient between them is 0.99.

4.2 ShortWalk Algorithm

Pair-wise combination gives equal weight for all nodes in the short traces. Thus, we can combine it with short random walks to generate the training pairs. Algorithm 1 describes our method. ShortWalk takes a graph $G = (V, E)$ as the input and initializes the SGNS model. It also initializes a walker that starts walking from a random node. At each step of a random walk, the walker randomly traverses to

Algorithm 1: ShortWalk algorithm

```
   Input : Graph G = (V, E); maximum walk length l;
           embedding size d; sampling budget S
 1 Initialize SGNS;
 2 while number of trained pair < S do
 3 |   currentNode = a random node from V;
 4 |   trace = (currentNode);
 5 |   while length(trace) < l & currentNode has neighbors do
 6 |   |   currentNode = a random neighbor of currentNode;
 7 |   |   append currentNode to trace;
 8 |   end
 9 |   for i = 0; i < len(trace); i++ do
10 |   |   for j = 0; j < len(trace); j++ do
11 |   |   |   if i ≠ j then
12 |   |   |   |   Update SGNS with (trace[i], trace[j])
13 |   |   |   end
14 |   |   end
15 |   end
16 end
```

one of the current location's neighbors. If the current location has no out-going edge, or the length of current trace excess the threshold l, it will teleport to a random node. Meanwhile, ShortWalk will generate the training pairs by taking the pair-wise combination of all nodes occurred in that trace to update SGNS. The algorithm stops when the number of training pairs been updated meets the preset sampling budget S.

The differences between ShortWalk and DeepWalk are: 1) ShortWalk uses a smaller l to generate the walking traces than DeepWalk. In most DeepWalk applications, l is set to a large value around 100. In ShortWalk, l is the highest proximity we want to reserve of the graph, which can be treated as the window size in DeepWalk. 2) ShortWalk generates the training pairs by taking pair-wise combination of all nodes occurred in the walking path (line 9,10 in Algorithm 1). Compared with DeepWalk, which uses SGNS to generate the training pairs, ShortWalk gives equal weight to all nodes in the same path.

5 Experiments

We evaluate our ShortWalk against DeepWalk (Perozzi et al., 2014). We do not compare with matrix factorization approaches since SGNS-based algorithms are proved to be superior. For SNGS based algorithms, most utilize additional attributes of the graph, and biased random walks that in favour of certain datasets. The closest to DeepWalk is Node2Vec. It is not compared with because the performance hinges on the tuning of hyper-parameters (p and q in the paper) data-wise. Besides, it is not scalable to large graphs even when the optimal hyper-parameter is found.

We experiment with two versions of DeepWalk, i.e., the traditional one with long walk length ($l = 100$, denoted as DeepWalk^{100}), and our improved version that is tailored for directed graph with shorter walk length ($l = 5$, denoted as DeepWalk^{5}). Our ShortWalk performs short random walk in the same way as

DeepWalk[5]. The difference is in the generation of training pairs. I.e., instead of and generates training pairs with pair-wise combination.

Once the training pairs are generated, they are fed into the popular word-embedding algorithm SGNS. For each method, we set the number of negative samples per training pair to 5, the dimension of embeddings to 100. The learning rate decays from 0.025 to 0.0001. These are the common settings for SGNS based algorithms (Mikolov et al., 2013a). However, the iteration time was rarely discussed in the previous works. In our work, we optimize the models with the same number of training pairs per dataset for a fair comparison. For each dataset, we set the sampling budget to $2 * 100 * |V| * 10$. Intuitively, it is the size of parameters of SGNS model multiplied by 10. The preliminary experiment suggests the model will converge and give a good result with this value. For DeepWalk, the window size is set to 5. Thus, the maximum walking path length l for ShortWalk is set to 5 to capture the same structures of the graphs.

For the sake of a fair comparison, we reimplement all algorithms in one platform so that they can run under exactly the same hyper-parameters. Cython from scratch. BLAS (Basic Linear Algebra Subprograms) is used to accelerate the vector computation. Our implementation can perform up to 5.8 million updates per second per thread. We also use multi-thread to boost the training speed. It is faster than most existing implementations (Mikolov et al., 2013a; Řeh ˚ uřek and Sojka, 2010). Experiments are conducted on a server with 24 cores and 256 GB memory. The source code is available online [1].

5.1 Datasets

We exhaustively tested on all the labelled directed graph that are available, in particular the directed labeled graphs used in the survey paper (Zhang et al., 2017). These datasets fall into three categories. Cora, CiteSeer, PubMed, Cora Citation, and AMinerV8 are citation networks extracted from digital libraries. Each node represents an academic paper and each directed link is a citation. Some papers also have label information indicating the corresponding research fields. These datasets are widely used to benchmark the embedding algorithms.

We also used three well-known directed graph from SNAP (Leskovec and Krevl, 2014) and KONECT (Kunegis, 2013). For example, Wiki Vote is a social network that contains the voting data of Wikipedia before January 2008. It is used by (Sun et al., 2018) to evaluate ATP which is a network embedding algorithm that can preserve the asymmetric transitivity.

PageRank is originally proposed to measure the importance of webpages. Thus we also experiment with two webpages datasets: WebGoogle and Web BerkStan. Webpages in WebGoogle are split by services. We use the two largest services (intl and univ) as the ground-true labels.

We clean the graphs and only use the largest weakly connected component (WCC) in our experiment. The statistics of the datasets are list in Table 2. The smallest dataset only contains 2,110 nodes and the largest one has over 0.77 million nodes linked by 4.18 million edges.

[1] `http://zhang18f.myweb.cs.uwindsor.ca/shortwalk`

Table 2 Statistics of datasets. We also list average shortest paths and number of triangles for smaller graphs to understand their structure. The average shortest path and number of triangles are not reported for AMinerV8 due to its large size.

Dataset	# Nodes	# Edges	Avg degree	Avg shortest path	# Triangles	# Labels
CiteSeer	2,110	3,757	1.78	1.52	1,083	6
Cora	2,485	5,209	2.10	4.57	1,558	7
wiki Vote	7,066	103,663	14.67	3.34	608,389	–
WebGoogle	15,763	171,206	10.86	6.33	591,156	2
PubMed	19,717	44,338	2.25	4.32	12,520	3
Cora Citation	23,166	91,500	3.95	13.82	78,791	10
Web BerkStan	654,782	7,499,425	11.45	13.75	64,520,617	–
AMinerV8	766,059	4,181,905	5.46	–	–	11

Table 3 Performance of classification task. Scores are averaged from 5 models. Each model produces one micro F1 score by 10-fold cross validation.

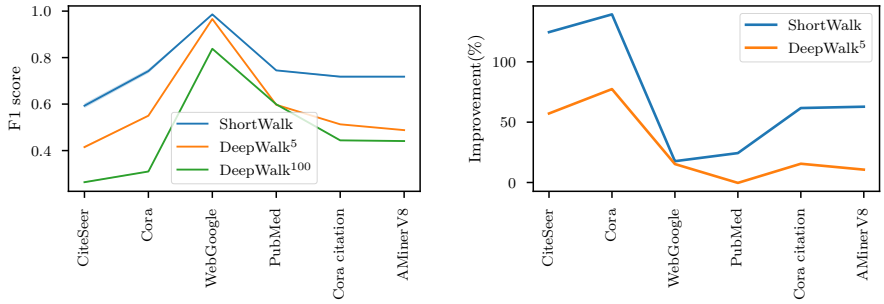
Dataset	DeepWalk^{100}	DeepWalk^{5}	ShortWalk
CiteSeer	0.264	0.415	0.593
Cora	0.310	0.550	0.742
WebGoogle	0.838	0.966	0.986
PubMed	0.599	0.597	0.745
Cora Citation	0.444	0.513	0.718
AMinerV8	0.441	0.488	0.718

5.2 Evaluation on Classification Task

Classification is widely used to evaluate network embeddings (Grover and Leskovec, 2016a; Tang et al., 2015b; Perozzi et al., 2014). Because of the stochastic nature of embedding algorithms, each run will produce a different embedding. To eliminate the effect of randomness, we run five models for each algorithm. Then for each model, we train a Logistic Regression classifier implemented in the scikit-learn toolkit with default hyper-parameters. The classifier takes an embedding as the input, then predicts the corresponding label of that node. We perform 10-fold cross-validation for each model and take the average micro F1 score as performance. Therefore, each model will have one performance score. Then we report the average and standard deviation of these five scores.

Table 3 and Figure 3 show the results, from which we have observations as follow:

1. Overall, ShortWalk outperforms DeepWalk^{100} and DeepWalk^{5} in classification task consistently. The highest performance is reported on WebGoogle. F1 for ShortWalk is 0.986. DeepWalk^{100} is 15% lower (0.838).
2. DeepWalk^{5} has better performance than DeepWalk^{100} on all datasets except PubMed. This indicates that shorter walking traces indeed can improve the quality of embeddings. However, the result is not satisfactory. For instance, the improvement is very small for large graphs such as PubMed, Cora Citation, and AMinerV8.
3. ShortWalk further improves DeepWalk^{5} by generating training pairs using the pair-wise combination. The improvement is significant. For example, Short-



(a) Average F1 score (b) Improvement over $DeepWalk^{100}$

Fig. 3 Performance of classification task. Each model produces one micro F1 score by 10-fold cross validation. Panel(a) reports the F1 score averaged from 5 models. The shaded area indicate the standard deviation. Panel(b) shows the corresponding improvement using $DeepWalk^{100}$ as the baseline.

Walk has 24.4%, 61.7%, 62.8% improvements against DeepWalk^5 in PubMed, Cora Citation, and AMinerV8.

4. Embeddings are stable in different runs. The standard deviation of the F1s is too small to observe in the plot. For instance, AMinerV8 has the smallest standard deviations of 0.003 and 0.001 for ShortWalk and DeepWalk.
5. The improvements vary for different datasets. The largest improvement of ShortWalk over DeepWalk^{100} is 139% in Cora. CiteSeer also receives 125% improvement. PubMed has only 315 labeled data. It has the smallest improvement among all datasets.

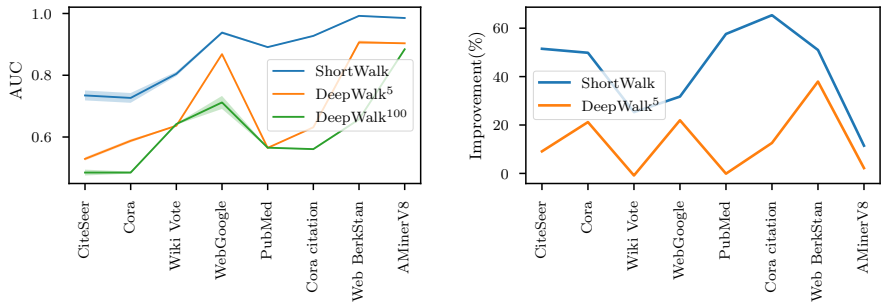
5.3 Evaluation on Link Prediction

In a graph, nodes interact with each other via links. Such links may be inaccurate or incomplete. Link prediction is a task to predict the missing links in a network (Liben-Nowell and Kleinberg, 2007). It is another popular benchmark for the evaluation of network embeddings (Grover and Leskovec, 2016a; Goyal and Ferrara, 2018b). In this task, each node has an embedding. Then the relation between two nodes can be represented by their embeddings using the *Hadamard* operator proposed in (Grover and Leskovec, 2016a).

To evaluate embeddings in this task, for each dataset, we use 70% proportion edges to learn embeddings and use the rest 30% edges as test data. In the evaluation phase, we treat the link prediction task as a regression task that calculates the probability of two nodes is linked by an edge in the network. Therefore, the true examples are the edges we removed before (the 30% proportion edges), and an equal amount of false examples are generated randomly. A Logistic Regression is used in this task. The output value is in the range of 0 to 1. Zero means very unlikely that two nodes are linked by an edge. One means these nodes are expected to be linked together. Then the performance is calculated by the Area Under the Curve (AUC) of the Receiver Operating Characteristic Curve (ROC) (Hanley and McNeil, 1982). This is the same strategy used in (Zhou et al., 2017; Goyal and

Table 4 AUC score of link prediction. 10-fold cross-validation. 5 embeddings per dataset per method.

Dataset	DeepWalk^{100}	DeepWalk^{5}	ShortWalk
CiteSeer	0.491	0.529	0.653
Cora	0.489	0.544	0.647
Wiki Vote	0.682	0.636	0.809
WebGoogle	0.718	0.868	0.934
PubMed	0.620	0.566	0.891
Cora Citation	0.596	0.631	0.927
Web BerkStan	0.664	0.905	0.993
AMinerV8	0.901	0.904	0.986



(a) Average AUC score (b) Improvement

Fig. 4 Performance of Link Prediction task. Each model produce one AUC score by 10-fold cross validation. Then the reported AUC score is averaged from 5 models. Panel(a) shows the AUC score of ShortWalk and DeepWalk. The shaded area indicate the standard deviation. Panel(b) shows the corresponding improvement using DeepWalk^{100} as the baseline.

Ferrara, 2018b). We run five models for each algorithm, then report the average AUC scores.

Table 4 and Figure 4 are the results. The x-axis shows the datasets sorted by the graph size. The y-axis denotes the AUC scores. Overall, ShortWalk outperforms DeepWalk^{100} and DeepWalk^{5} consistently on all datasets. Web BerkStan has the best performance in this task. The AUC scores are 0.99, 0.91, and 0.66 for ShortWalk, DeepWalk^{5}, and DeepWalk^{100}, respectively.

5.4 Discussions

ShortWalk improves DeepWalk from two aspects. It uses shorter walk traces as the 'text', and uses pair-wise combination to generate the training pairs. Next, we study two datasets to understand the impact of these two improvements.

We first take a look at the impact of the path length. Figure 5 shows the length of the traces retrieved by DeepWalk^{100} and DeepWalk^{5}. Panel (a) shows the WebGoogle dataset. When $l = 100$, the distribution of the trace length resembles a power-law. Most paths are short and few of them are long. However, we can see a peak at the end of the plot (length of 100). This is caused by self-loops in the directed graph. It contributes 9.17% of the total paths. When limiting the length

to 5, we can minimize the impact of these loops. This explains why DeepWalk^5 has higher performance than DeepWalk^{100} in both classification and link prediction tasks.

We plot the embeddings of in Figure 6. The first column shows the layout of the network generated by Atlas Force 2 (Jacomy et al., 2014). Columns 2, 3 and 4 are the embeddings generated by DeepWalk^{100}, DeepWalk^5, and ShortWalk, respectively. We use t-SNE (Van Der Maaten, 2014) to reduce the dimensionality from 100 to 2.

An interesting observation is that long random walks, demonstrated by $DeepWalk^{100}$, do generate long trails of nodes that belong to the same category. At the same time, there are large blobs of nodes from mixed categories that are threaded by the long walks. This will make classifiers difficult to separate them. Shorter walks in $DeepWalk^5$ reduce the size of the mixed blobs. Our ShortWalk can reduce break up the blobs further.

For PubMed data, the results of $DeepWalk^{100}$ and $DeepWalk^5$ are similar. A further investigation reveals that all the traces in this data are short. The longest length is only 11. This is because PubMed is a citation graph where papers only cite old ones. Hence, there is no loop in PubMed. It is expected to see that DeepWalk^{100} and DeepWalk^5 have similar performance in classification and link prediction tasks. Their 2D plots are also very similar. The only difference between DeepWalk^5 and ShortWalk is the way they generate the training pairs. As we discussed in Section 3.2, SGNS gives more weights to the nodes located in the center of the paths. By using pair-wise combination, each node on the same trace receives equal weighs, leading to better embeddings.

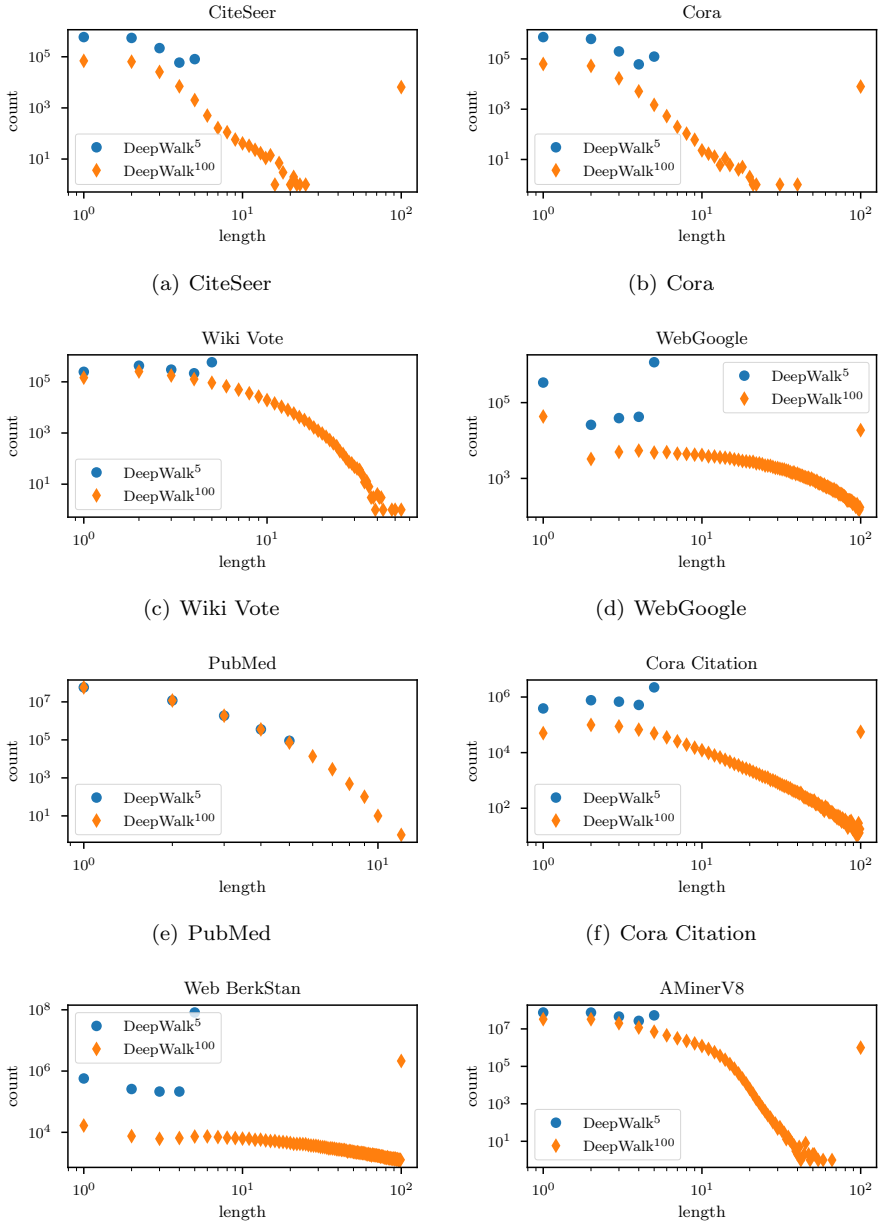
6 Conclusions

SGNS based network embedding algorithms are widely discussed and applied in real-world applications. However, these algorithms are designed for undirected graphs, where long random walks are used to capture the structure of the network. Applying these algorithms naively on directed graphs is problematic. This paper reveals two problems when applying random walk on directed graphs. Different from the undirected graph, long random walks can be trapped. Moreover, applying SGNS directly on these short traces will interrupt the node occurrence.

To overcome these problems, this paper proposes a novel but effective method called ShortWalk to learn embeddings from directed graphs. ShortWalk limits the length of random walk paths so that the impact of traps can be minimized. Instead of applying SGNS directly on the paths, we take the pair-wise combination to generate training pairs from the traces. This ensures all nodes in the same path will have equal weight during the learning process. We compare our approach with DeepWalk on 8 datasets. Experimental results show that ShortWalk outperforms DeepWalk consistently in both classification and link prediction tasks.

7 Acknowledgement

This research is supported by NSERC Discovery Grant RGPIN-2019-05350.



(a) CiteSeer (b) Cora

(c) Wiki Vote (d) WebGoogle

(e) PubMed (f) Cora Citation

(g) Web BerkStan (h) AMinerV8

Fig. 5 Length distribution of walk traces generated by DeepWalk with different walk length l.

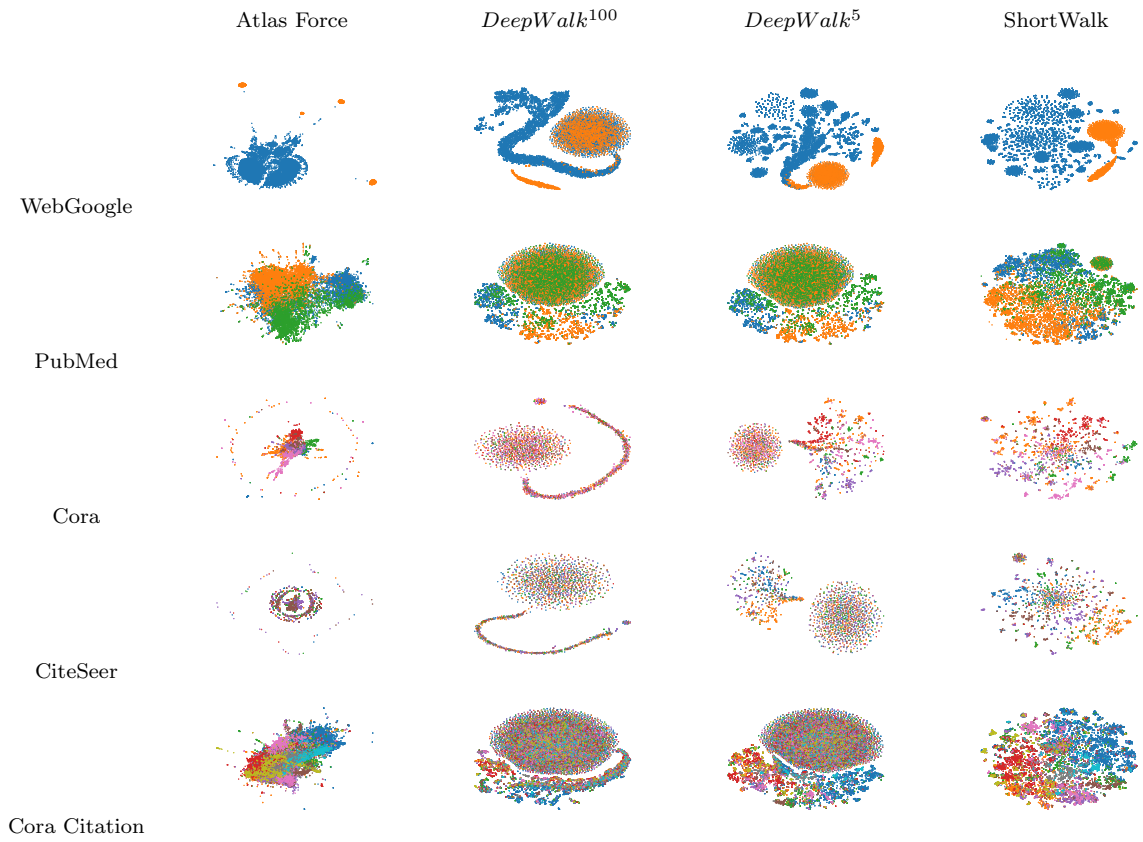


Fig. 6 2-D plot of embeddings generated by $DeepWalk^{100}$ (column 2), $DeepWalk^{5}$ (column 3), and *ShortWalk* (column 4). The embedding dimension is further reduced to two using t-SNE. The first column is generated using the Spring force network layout algorithm.

References

Sami Abu-El-Haija, Bryan Perozzi, and Rami Al-Rfou. 2017. Learning edge representations via low-rank asymmetric projections. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1787–1796.

Mikhail Belkin and Partha Niyogi. 2002. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*. 585–591.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3, Feb (2003), 1137–1155.

Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems* 30, 1-7 (April 1998), 107–117. `https://doi.org/10.1016/S0169-7552(98)00110-X`

Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

Shaosheng Cao, Wei Lu, and Qiongkai Xu. 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.

Mo Chen, Qiong Yang, Xiaoou Tang, et al. 2007. Directed Graph Embedding.. In *IJCAI*. 2707–2712.

Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. 2017a. Metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 135–144.

Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017b. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 135–144.

Palash Goyal and Emilio Ferrara. 2018a. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78–94.

Palash Goyal and Emilio Ferrara. 2018b. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowledge-Based Systems* 151 (2018), 78–94. https://doi.org/10.1016/j.knosys.2018.03.022

Vince Grolmusz. 2015. A Note on the PageRank of Undirected Graphs. *Inform. Process. Lett.* 115, 6-8 (June 2015), 633–634. https://doi.org/10.1016/j.ipl.2015.02.015

Aditya Grover and Jure Leskovec. 2016a. Node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 855–864.

Aditya Grover and Jure Leskovec. 2016b. node2vec: Scalable feature learning for networks. In *SIGKDD*. ACM, 855–864.

Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 297–304.

J A Hanley and B J McNeil. 1982. The Meaning and Use of the Area under a Receiver Operating Characteristic (ROC) Curve. *Radiology* 143, 1 (April 1982), 29–36. https://doi.org/10.1148/radiology.143.1.7063747

Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann, and Mathieu Bastian. 2014. ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software. *PLOS ONE* 9, 6 (June 2014), e98679. https://doi.org/10.1371/journal.pone.0098679

Megha Khosla, Jurek Leonhardt, Wolfgang Nejdl, and Avishek Anand. 2019. Node representation learning for directed graphs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 395–411.

Junghwan Kim, Haekyu Park, Ji-Eun Lee, and U Kang. 2018. Side: representation learning in signed directed networks. In *Proceedings of the 2018 World Wide Web Conference*. 509–518.

Jérôme Kunegis. 2013. KONECT: The Koblenz Network Collection. In *Proceedings of the 22Nd International Conference on World Wide Web (WWW '13 Companion)*. ACM, Rio de Janeiro, Brazil, 1343–1350. https://doi.org/10.1145/2487788.2488173

Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. (June 2014).

David Liben-Nowell and Jon Kleinberg. 2007. The Link-Prediction Problem for Social Networks. *Journal of the American Society for Information Science and Technology* 58, 7 (May 2007), 1019–1031. https://doi.org/10.1002/asi.20591

Kevin Lund and Curt Burgess. 1996. Producing High-Dimensional Semantic Spaces from Lexical Co-Occurrence. *Behavior Research Methods, Instruments, & Computers* 28, 2 (June 1996), 203–208. https://doi.org/10.3758/BF03204766

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013a. Distributed Representations of Words and Phrases and Their Compositionality. In *Advances in Neural Information Processing Systems*. 3111–3119.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.

Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric Transitivity Preserving Graph Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*. ACM Press, San Francisco, California, USA, 1105–1114. https://doi.org/10.1145/2939672.2939751

Gergely Palla, Illés J. Farkas, Péter Pollner, Imre Derényi, and Tamás Vicsek. 2007. Directed Network Modules. *New Journal of Physics* 9, 6 (June 2007), 186–186. https://doi.org/10.1088/1367-2630/9/6/186

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 701–710.

Bryan Perozzi, Vivek Kulkarni, and Steven Skiena. 2016. Walklets: Multiscale Graph Embeddings for Interpretable Network Classification. *arXiv preprint arXiv:1605.02115* (2016).

Radim Řeh ˚ uřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, 45–50.

T. Sen and D. K. Chaudhary. 2017. Contrastive Study of Simple PageRank, HITS and Weighted PageRank Algorithms: Review. In *2017 7th International Conference on Cloud Computing, Data Science Engineering - Confluence*. 721–727. https://doi.org/10.1109/CONFLUENCE.2017.7943245

Jiankai Sun, Bortik Bandyopadhyay, Armin Bashizade, Jiongqian Liang, P. Sadayappan, and Srinivasan Parthasarathy. 2018. ATP: Directed Graph Embedding with Asymmetric Transitivity Preservation. *CoRR* abs/1811.00839 (2018). arXiv:1811.00839

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015a. LINE: Large-scale information network embedding. In *WWW*. ACM, 1067–1077.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015b. Line: Large-Scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 1067–1077.

Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290, 5500 (2000), 2319–2323.

Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the 2018 World Wide Web Conference*. 539–548.

Laurens Van Der Maaten. 2014. Accelerating T-SNE Using Tree-Based Algorithms. *Journal of machine learning research* 15, 1 (2014), 3221–3245.

Yaojing Wang, Yuan Yao, Hanghang Tong, Feng Xu, and Jian Lu. 2020. A brief review of network embedding. *Big Data Mining and Analytics* 2, 1 (2020), 35–47.

Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2017. Network Representation Learning: A Survey. *arXiv:1801.05852 [cs, stat]* (Dec. 2017). arXiv:cs, stat/1801.05852

Fen Zhao, Yi Zhang, Jianguo Lu, and Ofer Shai. 2019. Measuring Academic Influence Using Heterogeneous Author-Citation Networks. *Scientometrics* 118, 3 (01 March 2019), 1119–1140. https://doi.org/10.1007/s11192-019-03010-5

Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable Graph Embedding for Asymmetric Proximity. In *Thirty-First AAAI Conference on Artificial Intelligence*.