Fast and Accurate Network Embeddings via Very Sparse Random Projection

Haochen Chen
haocchen@cs.stonybrook.edu
Stony Brook University

Syed Fahad Sultan
ssyedfahad@cs.stonybrook.edu
Stony Brook University

Yingtao Tian
yittian@cs.stonybrook.edu
Stony Brook University

Muhao Chen
muhaochen@ucla.edu
University of California, Los Angeles

Steven Skiena
skiena@cs.stonybrook.edu
Stony Brook University

ABSTRACT

We present FastRP, a scalable and performant algorithm for learning distributed node representations in a graph. FastRP is over 4,000 times faster than state-of-the-art methods such as DeepWalk and node2vec, while achieving comparable or even better performance as evaluated on several real-world networks on various downstream tasks. We observe that most network embedding methods consist of two components: construct a node similarity matrix and then apply dimension reduction techniques to this matrix. We show that the success of these methods should be attributed to the proper construction of this similarity matrix, rather than the dimension reduction method employed.

FastRP is proposed as a scalable algorithm for network embeddings. Two key features of FastRP are: 1) it explicitly constructs a node similarity matrix that captures transitive relationships in a graph and normalizes matrix entries based on node degrees; 2) it utilizes very sparse random projection, which is a scalable optimization-free method for dimension reduction. An extra benefit from combining these two design choices is that it allows the iterative computation of node embeddings so that the similarity matrix need not be explicitly constructed, which further speeds up FastRP. FastRP is also advantageous for its ease of implementation, parallelization and hyperparameter tuning. The source code is available at https://github.com/GTmac/FastRP.

CCS CONCEPTS

• **Information systems → Data mining**.

KEYWORDS

network embeddings; network representation learning; random projection

ACM Reference Format:
Haochen Chen, Syed Fahad Sultan, Yingtao Tian, Muhao Chen, and Steven Skiena. 2019. Fast and Accurate Network Embeddings via Very Sparse Random Projection. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM'19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3357879

1 INTRODUCTION

Network embedding methods learn low-dimensional distributed representation of nodes in a network. These learned representations can serve as latent features for a variety of inference tasks on graphs, such as node classification [23], link prediction [16] and network reconstruction [36].

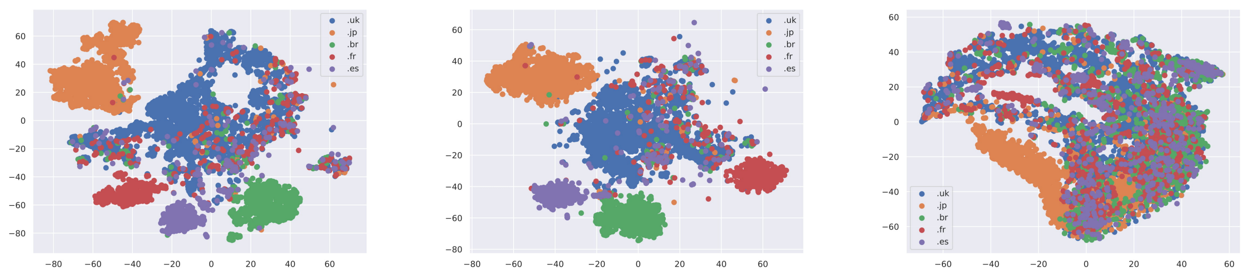
Research on network embeddings dates back to early 2000s in the context of dimension reduction, when methods such as LLE [27], IsoMap [32] and Laplacian Eigenmaps [4] were proposed. These methods are general in that they embed an arbitrary $n \times m$ feature matrix (n is the number of data points) into an $n \times d$ embedding matrix, where $d \ll m$. Although these methods produce high-quality embeddings, their time complexity is at least $O(n^2)$ which is prohibitive for large n.

More recent work in this area shifts their focus to embedding graph data, which represents a special class of sparse feature matrix, where $n = m$. The sparsity and discreteness of real-world graphs permit the design of more scalable network embedding algorithms. The pioneering work here is DeepWalk [23], which essentially samples node pairs from k-step transition matrices with different values of k, and then train a Skip-gram [22] model on these pairs to obtain node embeddings.

The most significant contribution of DeepWalk is that it introduces a two-component paradigm for representation learning on graphs: first explicitly constructing a node similarity matrix or implicitly sampling node pairs from it, then performing dimension reduction on the matrix to produce node embeddings. Much subsequent work has since followed to propose different strategies for both steps [16, 30, 33, 36].

Although most such methods are considered scalable with time complexity being linear to the number of nodes and/or edges, we note that the constant factor is often too large to be ignored. The reason is two-fold. First, many of these methods are sampling-based and a huge number of samples is required to learn high-quality embeddings. For example, DeepWalk samples about 32,000 context nodes per node under its default setting[1]. Second, the dimension reduction methods being used also incur substantial computational

[1]We consider the recommended hyperparameter settings in the DeepWalk paper, where 80 random walks of length 40 are sampled per node and the window size for Skip-gram is 10.



(a) FastRP. (b) DeepWalk. (c) RandNE.

Figure 1: Visualization of the embeddings produced by FastRP, DeepWalk and RandNE on the `WWW` network for websites from five country code top-level domains. We use t-SNE to project the embeddings to two-dimensional space.

cost, making this constant factor even larger. Popular methods such as DeepWalk [23], LINE [30] and node2vec [16] all adopt Skip-gram for learning node embeddings. However, optimizing the Skip-gram model is time-consuming due to the large number of gradient updates needed before the model converges. As such, despite being the most scalable state-of-the-art network embedding algorithms, it still takes DeepWalk and LINE several days of CPU time to embed the Youtube graph [31], a moderately sized graph with 1M nodes and 3M edges.

Can we design a truly scalable network embedding method that produces node embeddings for million-scale graphs in several minutes without compromising the representation quality? To answer this question, we analyze several state-of-the-art network embedding methods by examining their design choices for both similarity matrix construction and dimension reduction. Our analysis motivates us to propose FastRP, which presents much more scalable solutions to both steps without compromising embedding quality.

To illustrate the effectiveness of FastRP, we visualize the node representations produced by FastRP, DeepWalk [23] and an earlier random projection-based method, RandNE [38] on the WWW network (Figure 1). The nodes are hostnames such as `youtube.com` and `instagram.com`. For the purpose of visualization, we use t-SNE [21] to project the node embeddings to two-dimensional space. We take the websites from five countries: United Kingdom (`.uk`), Japan (`.jp`), Brazil (`.br`), France (`.fr`) and Spain (`.es`) as indicated by the color of the dots. Observe that for FastRP and DeepWalk, the websites from each top-level domain form clusters that are very well separated. For the RandNE embeddings, there is no clear boundary between the websites from different top-level domains. FastRP achieves similar quality to DeepWalk while being over 4,000 times faster.

To sum up, our contributions are the following:

Improved Understanding of Existing Network Embedding Algorithms. By viewing representative network embedding algorithms as a procedure with two components, similarity matrix construction and dimension reduction, we gain an improved understanding of why these algorithms work and why do they have scalability issues. This improved understanding motivates us to propose new solutions for both components.

Better Formulation of the Node Similarity Matrix. We construct a node similarity matrix with two unique properties: 1) it considers the implicit, transitive relationships between nodes; 2) it normalizes pairwise similarity of nodes based on node degrees.

More Scalable Dimension Reduction Algorithm. Different from previous work that relies on time-consuming dimension reduction methods such as Skip-gram and SVD, we obtain node embeddings via very sparse random projection of the node similarity matrix. An additional benefit from combining these two design choices is that it allows the iterative computation of node embeddings, which has linear cost in the size of the graph.

DeepWalk Quality Embeddings that is Produced Over 4,000 Times Faster. Extensive experimental results show that FastRP produces high-quality node embeddings comparable to state-of-the-art methods while being at least three orders of magnitude faster.

2 PRELIMINARIES

In this section, we give the formal definition of network embeddings and introduce the paradigm of network embeddings as a two-component process. We then detail the design decisions of several state-of-the-art methods for both components and show why they have scalability issues.

2.1 Notation and Task Definition

We consider the problem of embedding a network: given an undirected graph, the goal is to learn a low-dimensional latent representation for each node in the graph[2]. Formally, let $G = (V, E)$ be a graph, where V is the set of nodes and E is the set of edges. Let $n = |V|$ be the number of nodes, $m = |E|$ be the number of edges, d_i be the degree of the i-th node, and $\mathbf{S}$ be the adjacency matrix of G. The goal of network embeddings is to develop a mapping $\Phi : V \mapsto \mathbf{N} \in \mathbb{R}^{n\times d}, d \ll n$. For a node $v \in V$, we call the d-dimensional vector $\mathbf{N}_v$ its embedding vector (or node embedding).

Network embeddings can be viewed as performing *dimension reduction* on graphs: the input is an $n \times n$ feature matrix associated with the graph, on which we apply dimension reduction techniques to reduce its dimensionality to $n \times d$. This leads to two questions:

[2]We use network and graph interchangeably.

(1) What is an appropriate node similarity matrix to perform dimension reduction on?
(2) What dimension reduction techniques should be used?

We now review the existing solutions to both questions.

2.2 Node Similarity Matrix

The most straightforward input matrices to consider is the adjacency matrix S or the transition matrix A:

$$\mathbf{A} = \mathbf{D}^{-1}\mathbf{S}$$

where $\mathbf{D}$ is the degree matrix of G:

$$\mathbf{D}_{ij} = \begin{cases} \sum_p \mathbf{S}_{ip} & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

However, directly applying dimension reduction techniques on S or A is problematic. Real-world graphs are usually extremely sparse, which means most of the entries in S are zero. However, the absence of edge between two nodes u and v does not imply that there is no association between them. In particular, if two nodes are not adjacent but are connected by a large number of paths, it is still likely that there is a strong association between them.

The intuition above motivates us to exploit higher-order relationships in the graph. A natural high-order node similarity matrix is the k-step transition matrix:

$$\mathbf{A}^k = \underbrace{\mathbf{A} \cdots \mathbf{A}}_{k} \tag{1}$$

The ij-th entry of $\mathbf{A}^k$ denotes the probability of reaching j from i in exactly k steps of random walk. We will show that many existing methods adopt variations of this definition of similarity matrix.

2.3 Dimension Reduction Techniques

Once an $n \times n$ similarity matrix is constructed, network embedding methods perform dimension reduction on it to obtain node representations. In this section, we introduce two commonly used dimension reduction techniques: singular value decomposition (SVD) and Skip-gram.

SVD. SVD [17] is a classical matrix factorization method for dimension reduction. SVD factorizes a feature matrix M into the product of three matrices: $\mathbf{M} = \mathbf{U} \cdot \Sigma \cdot \mathbf{V}^\top$, where U and V are orthonormal and Σ is a diagonal matrix consisting of singular values. To perform dimension reduction with SVD, it is common to take the top d singular values Σ_d from Σ and the corresponding columns from U and V.

Skip-gram. Skip-gram [22] is a method for learning word embeddings, which is also shown to be performant in the context of network embeddings. Skip-gram works by sampling word pairs (or node pairs) from a word co-occurrence matrix (or node similarity matrix) C and modeling the probability that a given word-context pair (u, v) is from C or not. Goldberg and Levy [19] showed that Skip-gram is implicitly factorizing a shifted pointwise mutual information (PMI) matrix of word co-occurrences. Formally, the matrix Skip-gram seeks to factorize has elements:

$$\mathbf{M}_{uv} = \log \frac{\#(u, v) \cdot |C|}{\#(u) \cdot \#(v)} - \log b = \log(\text{PMI}(u, v)) - \log b \tag{2}$$

where $\#(u)$ denotes the occurrence count of u in C and b denotes the number of negative samples in Skip-gram.

2.4 Representative Network Embedding Methods

In this section, we discuss how three representative network embedding methods: DeepWalk [23], LINE [30] and GraRep [7] fit into the two-component procedure described above. The analysis of node2vec [16] is similar to that of DeepWalk, which we omit here.

DeepWalk [23]. DeepWalk's core idea is to sample node pairs from a weighted combination of $A, A^2, \cdots, A^k$, and then train a Skip-gram model on these samples. Making use of Eq. 2, it can be shown that DeepWalk is implicitly factorizing the following matrix [26]:

$$\mathbf{M} = \log\left(\text{vol}(G)\left(\frac{1}{k}\sum_{r=1}^{k} \mathbf{A}^r \mathbf{D}^{-1}\right)\right) - \log b \tag{3}$$

where $\text{vol}(G) = \sum_i \sum_j S_{ij}$.

LINE [30]. LINE can be seen as a variation of DeepWalk that only considers node pairs that are at most two hops away. Using a similar derivation, it can be shown that LINE implicitly factorizes:

$$\mathbf{M} = \log\left(\text{vol}(G)\left(\mathbf{A}\mathbf{D}^{-1}\right)\right) - \log b \tag{4}$$

GraRep [7]. GraRep can be regarded as the matrix factorization version of DeepWalk. Instead of sampling from $A, A^2, \cdots, A^k$, it directly computes these matrices and then factorizes the corresponding shifted PMI matrix for each power of A.

2.5 Scalability of Representative Methods

Putting existing methods into this two-component framework reveals their intrinsic scalability issues as following:

Node Similarity Matrix Construction. Many previous studies have demonstrated the importance of preserving high-order proximity between nodes [7, 23, 26, 38, 39], which is typically done by raising $\mathbf{A}$ to k-th power and optionally normalize it afterward (see Eq. 3 for an example). This causes scalability issues since both computing $\mathbf{A}^k$ and applying a transformation to each element in $\mathbf{A}^k$ are at least quadratic. For methods such as DeepWalk and node2vec, this problem is slightly mitigated by sampling node pairs from $\mathbf{A}^k$ instead. But still, a huge number of samples is required for them to get an accurate enough estimation of $\mathbf{A}^k$.

Dimension Reduction. The dimension reduction techniques adopted by these methods also affect their scalability. Both Skip-gram and SVD are not among the fastest dimension reduction algorithms [35].

In the next section, we present our solutions to both problems that allow for better scalability.

3 METHOD

In this section, we introduce FastRP. We first describe the usage of very sparse random projection for dimension reduction and its merit in preserving high-order proximity. Then, we present our design of the node similarity matrix. This matrix is carefully designed so that: 1) it preserves transitive relationships in the input graph; 2) its entries are properly normalized; 3) it can be formulated as matrix chain multiplication, so that applying random projection on this matrix only costs linear time. Lastly, we discuss several additional advantages of FastRP.

3.1 Very Sparse Random Projection

Random projection is a dimension reduction method that preserves pairwise distances between data points with strong theoretical guarantees [35]. The idea behind this is very simple: to reduce an $n \times m$ (for graph data, we have $n = m$) feature matrix $\mathbf{M}$ to an $n \times d$ matrix $\mathbf{N}$ where $d \ll m$, we can simply multiply the feature matrix with an $m \times d$ random projection matrix $\mathbf{R}$:

$$\mathbf{N} = \mathbf{M} \cdot \mathbf{R} \tag{5}$$

As long as the entries of $\mathbf{R}$ are i.i.d with zero mean, $\mathbf{N}$ is able to preserve the pairwise distances in $\mathbf{A}\mathbf{A}^\top$ [2].

The difference among different random projection algorithms is mostly in the construction of $\mathbf{R}$. The most studied one is Gaussian random projection, where entries of $\mathbf{R}$ are sampled i.i.d. from a Gaussian distribution: $\mathbf{R}_{ij} \sim \mathcal{N}(0, 1/d)$. Since $\mathbf{R}$ is a dense $m \times d$ matrix, the time complexity of Gaussian random projection is $O(n \cdot m \cdot d)$.

As an improvement to Gaussian random projection, Achlioptas [1] proposed *sparse random projection*, where entries of $\mathbf{R}$ are sampled i.i.d. from

$$\mathbf{R}_{ij} = \begin{cases} \sqrt{s} \text{ with probability } \dfrac{1}{2s} \\ 0 \text{ with probability } 1 - \dfrac{1}{s} \\ -\sqrt{s} \text{ with probability } \dfrac{1}{2s} \end{cases} \tag{6}$$

where $s = 3$ is used. This leads to a 3x speedup since 2/3 of the entries of $\mathbf{R}$ are zero. Additionally, this configuration of $\mathbf{R}$ does not require any floating-point computation since the multiplication with $\sqrt{s}$ can be delayed, providing additional speedup.

Li et al. [20] extend Achlioptas [1] by showing that $s \gg 3$ can be used to further speed up the computation. They recommend setting $s = \sqrt{m}$, which achieves $\sqrt{m}$ times speedup over Gaussian random projection while ensuring the quality of the embeddings. In this work, we consider this *very sparse random projection* method for dimension reduction of the node similarity matrix.

As an optimization-free dimension reduction method, very sparse random projection wins over SVD and Skip-gram for its superior computational efficiency. The fact that it only requires matrix multiplication also enables faster computation on accelerators such as GPUs, as well as easy parallelization.

Apart from these advantages, the random projection approach also benefits from the *associative* property of matrix multiplication. To see why this is important, consider the basic form of high-order similarity matrix $\mathbf{A}^k$, as defined in Eq. 1. To compute its random projection $\mathbf{N} = \mathbf{A}^k \cdot \mathbf{R}$, there is no need to calculate $\mathbf{A}^k$ from scratch since the computation can be done iteratively:

$$\mathbf{N} = (\underbrace{\mathbf{A} \cdot \dots \cdot (\mathbf{A} \cdot (\mathbf{A}}_{k} \cdot \mathbf{R}))) \tag{7}$$

This reduces the time complexity from $O(n^3 \cdot k \cdot d)$ to $O(m \cdot k \cdot d)$.

3.2 Similarity Matrix Construction

The next step is to construct a proper node similarity matrix leveraging the associative property of matrix multiplication. We make two key observation about the similarity matrices used by the existing method. First, it is important to preserve high-order proximity in the input graph – this is typically done by raising $\mathbf{A}$ to k-th power. Second, element-wise normalization is performed (taking logarithm for DeepWalk, LINE and GraRep) on the raw similarity matrix before dimension reduction.

Most previous matrix-based network embedding methods emphasize on the importance of high-order proximity but skip the element-wise normalization step for either better scalability or ease of analysis [25, 26, 38]. Is normalization of the node similarity matrix important? If so, is there any other normalization method that allows for scalable computation? We answer these questions by analyzing the properties of $\mathbf{A}^k$ from a spectral graph theory perspective.

To begin with, we consider a transformation of $\mathbf{A}$ defined as $\mathbf{B} = \mathbf{D}^{\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$. Since $\mathbf{B}$ is a real symmetric matrix, it can be decomposed as $\mathbf{B} = \mathbf{Q}\Lambda\mathbf{Q}^\top$ where Λ is a diagonal matrix of eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ of $\mathbf{B}$, and $\mathbf{Q}$ is an orthogonal matrix consisting of the corresponding eigenvectors $\mathbf{q}_1, \ldots, \mathbf{q}_n$.

It is easy to verify that $\left(1, \mathbf{w} = \left(\sqrt{d_1}, \cdots, \sqrt{d_n}\right)\right)$ is an eigenpair of $\mathbf{B}$. Following the Frobenius-Perron Theorem [15], we have:

$$\lambda_1 = 1 > \lambda_2 \geq \ldots \geq \lambda_n \geq -1$$

and

$$\mathbf{q}_1 = \left(\sqrt{\frac{d_1}{2m}}, \cdots, \sqrt{\frac{d_n}{2m}}\right)$$

Now:

$$\begin{aligned} \mathbf{A}^k &= \mathbf{D}^{-\frac{1}{2}}\mathbf{B}^k\mathbf{D}^{\frac{1}{2}} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{Q}\Lambda\mathbf{Q}^\top)^k\mathbf{D}^{\frac{1}{2}} \\ &= \mathbf{D}^{-\frac{1}{2}}\mathbf{Q}\Lambda^k\mathbf{Q}^\top\mathbf{D}^{\frac{1}{2}} \\ &= \sum_{t=1}^{n} \lambda_t^k \mathbf{D}^{-\frac{1}{2}}\mathbf{q}_t\mathbf{q}_t^\top\mathbf{D}^{\frac{1}{2}} \\ &= \mathbf{P} + \sum_{t=2}^{n} \lambda_t^k \mathbf{D}^{-\frac{1}{2}}\mathbf{q}_t\mathbf{q}_t^\top\mathbf{D}^{\frac{1}{2}} \end{aligned}$$

where $\mathbf{P}_{ij} = d_j/2m$.

For a particular entry $\mathbf{A}^k_{ij}$ we have:

$$\mathbf{A}^k_{ij} = \frac{d_j}{2m} + \sum_{t=2}^{n} \lambda_t^k q_{ti} q_{tj} \sqrt{\frac{d_j}{d_i}} \tag{8}$$

This derivation illustrates the importance of normalization. Since $|\lambda_t| < 1$ holds for $t = 2, \ldots, n$ (assuming G is non-bipartite), we have $\mathbf{A}^k_{ij} \to d_j/2m$ when $k \to \infty$. Since many of the real-world graphs are scale-free [3], it follows that the entries in $\mathbf{A}^k$ also has a heavy-tailed distribution.

The heavy-tailed distribution of data causes problems for dimension reduction methods [20]. The pairwise distances between data points are dominated by the columns with exceptionally large values, rending them less meaningful. In practice, term weighting schemes are applied to heavy-tailed data to reduce its kurtosis and skewness [28]. Here, we consider a scaled version of the Tukey

Algorithm 1 FastRP(A)

```
Input:
    graph transition matrix A, embedding dimensionality $d$, maximum power $k$, normalization strength $\beta$, weights $\alpha_1, \alpha_2, \ldots, \alpha_k$
Output: matrix of node representations $\mathbf{N} \in \mathbb{R}^{n \times d}$
1: Produce $\mathbf{R} \in \mathbb{R}^{n \times d}$ according to Eq. 6
2: $\mathbf{N}_1 \leftarrow \mathbf{A} \cdot \mathbf{L} \cdot \mathbf{R}$ where $\mathbf{L}_{ij} = \left(\frac{d_j}{2m}\right)^{\beta}$
3: for $i = 2$ to $n$ do
4:     $\mathbf{N}_i \leftarrow \mathbf{A} \cdot \mathbf{N}_{i-1}$
5: end for
6: $\mathbf{N} = \alpha_1 \mathbf{N}_1 + \ldots + \alpha_k \mathbf{N}_k$
7: return N
```

transformation [34]. Concretely, we transform a feature y into y^{λ}, where λ controls the strength of normalization. Now the only problem is that the exact feature values in $\mathbf{A}^k$ are not known, and we do not want to calculate these values for better scalability. But again, we can rely on the fact that $\mathbf{A}^k_{ij}$ converges to $d_j/2m$. The normalization we consider is therefore:

$$\tilde{\mathbf{A}}^k_{ij} = \mathbf{A}^k_{ij} \cdot \left(\frac{d_j}{2m}\right)^{\lambda-1} \approx \mathbf{A}^k_{ij} \cdot \left(\mathbf{A}^k_{ij}\right)^{\lambda-1} \approx \left(\mathbf{A}^k_{ij}\right)^{\lambda} \tag{9}$$

3.3 Our Algorithm: FastRP

Let $\beta = \lambda - 1$, the normalization scheme in Eq. 9 can be represented in matrix form: $\tilde{\mathbf{A}}^k = \mathbf{A}^k \cdot \mathbf{L}$ where $\mathbf{L} = \operatorname{diag}\left(\left(\frac{d_1}{2m}\right)^{\beta}, \ldots, \left(\frac{d_n}{2m}\right)^{\beta}\right)$. This allows for matrix chain multiplication when performing random projection:

$$\mathbf{N} = \tilde{\mathbf{A}}^k \cdot \mathbf{R} = (\underbrace{\mathbf{A} \cdots (\mathbf{A} \cdot (\mathbf{A}}_{k} \cdot \mathbf{L} \cdot \mathbf{R})))$$

We further consider a weighted combination of different powers of $\mathbf{A}$, so that the embeddings of G is computed as follows:

$$\mathbf{N} = \left(\alpha_1 \tilde{\mathbf{A}} + \alpha_2 \tilde{\mathbf{A}}^2 + \ldots + \alpha_k \tilde{\mathbf{A}}^k\right) \cdot \mathbf{R}$$

where $\alpha_1, \alpha_2, \ldots, \alpha_k$ are the weights. The outline of FastRP is presented in Algorithm 1.

3.4 Time Complexity

The time complexity of FastRP is $O((n \cdot d)/s) = O\left(n \cdot \sqrt{d}\right)$ for constructing the sparse random projection matrix (line 1), $O(m \cdot k \cdot d)$ for random projection (line 2 to 5) for each power of $\mathbf{A}$ and $O(n \cdot k \cdot d)$ for merging embedding matrices (line 6). Overall, the time complexity of FastRP is $O((n + m) \cdot k \cdot d)$, which is linear to the number of nodes and edges in G.

3.5 Implementation, Parallelization and Hyperparameter Tuning

Implementation. The implementation of FastRP is very simple and straightforward, with less than 100 lines of Python code.

Parallelization. Besides the ease of implementation, our algorithm is also easy to parallelize, since the only operation involved is matrix multiplication. One easy way to speed up matrix multiplication $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$ is to perform block partitioning on the input matrices $\mathbf{A}$ and $\mathbf{B}$:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \tag{10}$$

Then it is easy to see that:

$$C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix} \tag{11}$$

The recursive matrix multiplications and summations can be performed in parallel. The smaller block matrices $A_{11}, A_{12}, \cdots, B_{11}, B_{12}, \cdots$ can be further partitioned recursively for execution on more processors [5].

Efficient Hyperparameter Tuning. FastRP also allows for highly efficient hyperparameter tuning. The idea is to first pre-compute the embeddings $\mathbf{N}_1, \mathbf{N}_2, \cdots, \mathbf{N}_k$ derived from different orders of proximity matrices. Since the final embedding matrix is a weighted combination of $\mathbf{N}_1, \mathbf{N}_2, \cdots, \mathbf{N}_k$, we only need to perform weighted summation during hyperparameter optimization. Furthermore, according to the Johnson-Lindenstrauss lemma [12], the embedding dimensionality d determines the approximation error of random projection. The implication of this is that we can efficiently tune hyperparameters on smaller values of d. This is in contrast to most of the existing algorithms, which require retraining of the entire model for each hyperparameter configuration.

Besides these merits, the biggest advantage of FastRP is its superior computational efficiency. In the next section, we will show that FastRP is orders of magnitude faster than the state-of-the-art methods while achieving comparable or even better performance.

4 EXPERIMENTS

In this section, we conduct experiments to evaluate the performance of FastRP. We first provide an overview of the datasets. Then, we compare FastRP with a number of baseline methods both in terms of running time and performance on downstream tasks. We further discuss the performance of our method with regard to several important hyperparameters and its scalability.

4.1 Datasets

Table 1 gives an overview of the datasets used in experiments.

- WWW-200K and WWW-10K [11]: these graphs are derived from the Web graph provided by Common Crawl, where the nodes are hostnames and the edges are the hyperlinks between these websites. For simplicity, we treat this graph as an undirected graph. The original graph has 385 million nodes and 2.5 billion edges, which is too large to be loaded into the memory of our machine. Thus, we construct subgraphs of this graph by taking the top 200,000 and 10,000 websites respectively as ranked by Harmonic Centrality [6]. We also use the `WWW-10K` graph for node classification, for which the label of a node is its top-level domain name such as `.org`, `.edu` and `.es`.
- Blogcatalog [31]: this is a network between bloggers on the Blogcatalog website. The labels are the categories a blogger publishes in.

Name	# Vertices	# Edges	# Classes	Task
WWW-200K	200,000	32,822,166	-	K-Nearest Neighbors
WWW-10K	10,000	3,904,610	50	Node Classification
Blogcatalog	10,312	333,983	39	Node Classification
Flickr	80,513	5,899,882	195	Node Classification

Table 1: Statistics of the graphs used in our experiments.

- Flickr [31]: this is a network between the users on the photo sharing website Flickr. The labels represent the interest groups a user joins.

4.2 Baseline Methods

We compare FastRP against the following baseline methods:

- DeepWalk [23] – DeepWalk is a network embedding method that samples short random walks from the input graph. Then, these random walks are fed into the Skip-gram model to produce node embeddings.
- node2vec [16] – node2vec extends DeepWalk by performing biased random walks that balance between DFS and BFS. It also adopts SGNS as the dimension reduction method.
- LINE [30] – LINE samples node pairs that are either adjacent or two hops away. LINE adopts Skip-gram with negative sampling (SGNS) to learn network embeddings from the samples.
- RandNE [38] – RandNE constructs a node similarity matrix that preserves the high-order proximity by raising the adjacency (or transition) matrix to the k-th power. Then, node embeddings are obtained by applying Gaussian random projection to this matrix.

We realize that there are many other recently proposed network embedding methods. We do not include these methods as baselines since their performance are generally inferior to DeepWalk according to a recent comparative study [18]. Moreover, many of them are not scalable [18].

4.3 Parameter Settings

Here we present the parameter settings for the baseline models and our model.

FastRP . For FastRP, we set embedding dimensionality d to 512 and maximum power k to 4. For the weights $\alpha_1, \alpha_2, \cdots, \alpha_k$, we observe that simply use a weighted combination of $\mathbf{A}^3$ and $\mathbf{A}^4$ is already enough for achieving competitive results. Thus, we set $\alpha_1, \alpha_2, \alpha_3$ to $0, 0, 1$ respectively and tune α_4.

Overall, we only have two hyperparameters to tune: normalization strength β and the weight α_4 for $\mathbf{A}^4$. We use optuna[3], a Bayesian hyperparameter optimization framework to tune them. The hyperparameter optimization is performed for 20 rounds on a small validation set of 1% labeled data; the search ranges for β and α_4 are set to $[-1, 0]$ and $[2^{-3}, 2^6]$ respectively. We also use a lower embedding dimensionality of $d = 64$ to speed up the tuning process.

[3]https://github.com/pfnet/optuna

RandNE. For RandNE, we set embedding dimensionality d to 512 and maximum order q to 3. We note that for RandNE, incorporating the embeddings from $\mathbf{A}^4$ does not improve the quality of embeddings according to our experiments. To ensure a fair comparison, we also conduct hyperparameter search for the weights in RandNE using the same procedure as FastRP. The only difference is that instead of tuning β and α_4, we optimize the weights of $\mathbf{A}^2$ and $\mathbf{A}^3$.

DeepWalk. For DeepWalk, we need to set the following parameters: the number of random walks γ, walk length t, window size w for the Skip-gram model and representation size d. We adopt the hyperparameter settings recommended in the original paper: $\gamma = 80, t = 40, w = 10, d = 128$.

node2vec. Since node2vec is built upon DeepWalk, we use the same parameter settings for node2vec as DeepWalk: $\gamma = 80, t = 40, w = 10, d = 128$. We notice that these parameter settings lead to better results than the default settings as described in the paper, possibly because the total number of samples is larger. For the in-out parameter p and return parameter q, we conduct grid search over $p, q \in \{0.25, 0.50, 1, 2, 4\}$ as suggested in the paper.

LINE. We use LINE with both the first order and second order proximity with the recommended hyperparameters. Concretely, we set the dimensionality of embeddings to 200, the number of node pair samples to 10 billion and the number of negative samples to 5.

All the experiments are conducted on a single machine with 128 GB memory and 40 CPU cores at 2.2 GHz. We note that FastRP and all the baseline methods support multi-threading. However, for a fair running time comparison, we run all methods with a single thread and measure the CPU time (process time) consumed by each method.

4.4 Runtime Comparison

We first showcase the superior efficiency of our method by reporting the CPU time of FastRP and the baseline methods on all datasets in Table 2. FastRP achieves at least 4,000x speedup over the state-of-the-art method DeepWalk. For example, it takes FastRP less than 3 minutes to embed the `WWW-200K` graph, whereas DeepWalk takes almost a week to finish. Node2vec is even slower; although LINE is several times faster than DeepWalk and node2vec, it is still a few hundreds of times slower than FastRP. The only method with comparable running time is RandNE which uses Gaussian random projection for dimension reduction, but it is also slightly slower than FastRP. Moreover, in the experiments below, we will show that the quality of embeddings produced by FastRP is significantly better than that of RandNE.

Dataset	**Algorithm**					Speedup over
	FastRP	RandNE	LINE	DeepWalk	node2vec	DeepWalk
`WWW-200K`	136.0 seconds	169.8 seconds	4.6 hours	6.9 days	63.8 days	**4383x**
`WWW-10K`	7.8 seconds	13.6 seconds	3.2 hours	9.2 hours	59.8 hours	**4246x**
`Blogcatalog`	6.0 seconds	10.5 seconds	3.0 hours	8.7 hours	41.2 hours	**5220x**
`Flickr`	33.1 seconds	45.1 seconds	4.2 hours	3.1 days	28.5 days	**8091x**

Table 2: CPU time comparison on all test datasets. FastRP is over 4,000 times faster than the state-of-the-art algorithm DeepWalk.

Table 3: Top 5 nearest neighbors of four representative websites calculated from the node embeddings generated by FastRP, DeepWalk and RandNE respectively. Rows are arranged from the highest cosine similarity to lowest cosine similarity.

Methods	`FastRP`	`DeepWalk`	`RandNE`	`FastRP`	`DeepWalk`	`RandNE`
Websites		`nytimes.com`			`delta.com`	
Neighbors	huffingtonpost.com	washingtonpost.com	huffingtonpost.com	aa.com	aa.com	aa.com
	washingtonpost.com	huffingtonpost.com	washingtonpost.com	united.com	united.com	southwest.com
	cnn.com	cnn.com	forbes.com	usairways.com	usairways.com	united.com
	npr.org	cbsnews.com	cnn.com	alaskaair.com	southwest.com	expedia.com
	latimes.com	time.com	npr.org	jetblue.com	jetblue.com	priceline.com
Methods	`FastRP`	`DeepWalk`	`RandNE`	`FastRP`	`DeepWalk`	`RandNE`
Websites		`vldb.org`			`arsenal.com`	
Neighbors	sigmod.org	sigmod.org	comp.nus.edu.sg	chelseafc.com	chelseafc.com	liverpoolfc.com
	comp.nus.edu.sg	morganclaypool.com	cs.sfu.ca	mcfc.co.uk	tottenhamhotspur.com	manutd.com
	sigops.org	kdd.org	cs.rpi.edu	nufc.co.uk	manutd.com	chelseafc.com
	cidrdb.org	doi.acm.org	nlp.stanford.edu	avfc.co.uk	mcfc.co.uk	skysports.com
	cse.iitb.ac.in	informatic.uni-trier.de	theory.stanford.edu	tottenhamhotspur.com	thefa.com	tottenhamhotspur.com

4.5 Qualitative Case Study: WWW-200K Network

We first conduct a case study on the WWW-200K network to compare the embeddings produced by different network embedding algorithms qualitatively. For this part, we take RandNE and DeepWalk as the baselines since the results produced by LINE and node2vec are very similar to those of DeepWalk on this network.

Examining the K-nearest neighbors (KNN) of a word is a common way to measure the quality of word embeddings [22]. In the same spirit, we examine the K-nearest neighbors of several representative websites in the node embedding space. Cosine similarity is used as the similarity metric. Table 3 lists the top 5 nearest neighbors of four representative websites: `nytimes.com`, `delta.com`, `vldb.org`, and `arsenal.com` based on the node embeddings produced by FastRP, RandNE and DeepWalk.

`nytimes.com` is the website of The New York Times, which is one of the most influential news websites in the US. We find that all three methods produce high-quality nearest neighbors for `nytimes.com`: `huffingtonpost.com`, `washingtonpost.com` and `cnn.com` are also well-known American news sites. It is also interesting to see that FastRP lists `latimes.com` (The Los Angeles Times) among the top 5 nearest neighbors of `nytimes.com`.

`delta.com` is the homepage of Delta Air Lines, which is a major American airline. Again, we find that the most similar websites discovered by FastRP and DeepWalk are the official websites of other major American airlines: American Airlines (`aa.com` and `usairways.com`), United Airlines (`united.com`), Alaska Airlines (`alaskaair.com`), etc. The nearest neighbors list provided by RandNE is worse, since it includes general purpose travel websites such as `expedia.com` and `priceline.com`.

`vldb.org` is the official website for VLDB Endowment, which steers the VLDB conference, a leading database research conference. Both FastRP and DeepWalk list `sigmod.org` as the most similar website to `vldb.org`; this is a positive sign since SIGMOD is another top database research conference. On the other hand, all three methods include several universities' CS department websites in the nearest neighbors list, such as `comp.nus.edu.sg` and `cs.sfu.ca`. In particular, all top five websites provided by RandNE are CS department websites. In our opinion, this is reasonable but less satisfactory than having other CS research conferences' websites in the list, such as `sigops.org` (The ACM Special Interest Group in Operating Systems) and `cidrdb.org` (The Conference on Innovative Data Systems Research).

`arsenal.com` represents a football club that plays in the Premier League. It can be seen that all three methods list the other football clubs in the Premier League as the nearest neighbors of `arsenal.com`, such as Chelsea (`chelseafc.com`) and Manchester City (`mcfc.co.uk`). The only exception is RandNE, which also lists skysports.com (the dominant subscription TV sports brand in the United Kingdom) as one of the nearest neighbors. skysports.com has higher popularity but is less relevant to arsenal.com. In contrast, FastRP avoids this problem by properly downweights the influence of popular nodes. Overall, we find that the quality of nearest neighbors produced by FastRP is comparable to DeepWalk and significantly better than that of RandNE.

The experiments above serve as a qualitative evaluation of FastRP. In the following sections, we will conduct quantitative evaluations of FastRP on different downstream tasks across multiple datasets.

Algorithm	**Dataset**		
	WWW-10K	BlogCatalog	Flickr
LINE	6.66	19.63	10.69
node2vec	**27.42**	21.44	11.89
DeepWalk	25.54	21.30	14.00
RandNE	15.68	20.88	13.64
FastRP	26.92	**23.43**	**15.02**

Table 4: Macro F_1 scores of all methods on WWW-10K, BlogCatalog, and Flickr in percentage (Section 4.6).

4.6 Multi-label Node Classification

For the task of node classification, we evaluate our method using the same experimental setup in DeepWalk [23]. Firstly, a portion of nodes along with their labels are randomly sampled from the graph as training data, and the goal is to predict the labels for the rest of the nodes. Then, a one-vs-rest logistic regression model with L2 regularization is trained on the node embeddings for prediction. We use the logistic regression model implemented by LibLinear [13]. To ensure the reliability of our experiment, the above process is repeated for 10 times, and the average Macro F_1 score is reported. The other evaluation metrics such as Micro F_1 score and accuracy follow the same trend as Macro F_1 score, thus are not shown.

Table 4 reports the Macro F_1 scores achieved on WWW-10K, Blogcatalog and Flickr with 1%, 10% and 1% labeled nodes respectively. FastRP achieves the state-of-the-art result on two out of three datasets and matches on the third. On Blogcatalog and Flickr, FastRP achieves gains of 9.3% and 7.3% over the best performing baseline method respectively. On WWW-10K, the absolute difference in F_1 score between FastRP and node2vec is only 0.5%, but FastRP is over 10,000 times faster.

To have a detailed comparison between FastRP and the baseline methods, we vary the portion of labeled nodes for classification and present the macro F_1 scores in Figure 2. We can observe that FastRP consistently outperforms or matches the other neural baseline methods, while being at least three orders of magnitude faster. It is also clear that FastRP always outperforms RandNE by a large margin, proving the effectiveness of our node similarity matrix design.

4.7 Parameter Sensitivity

To examine how do the hyperparameters affect the quality of learned representations, we conduct a parameter sensitivity study on BlogCatalog. The parameters we investigate are the normalization strength β, $\mathbf{A}^4$'s weight α_4, and the embedding dimensionality d. We report the Macro F_1 score achieved with 10% labeled data in Figure 3.

Normalization strength. Figure 3a shows the effectiveness of our normalization scheme. At $\beta = -0.9$, FastRP achieves the highest F_1 score of over 20.6%. By setting β to 0.0, no normalization is applied to the node similarity matrix and the F_1 score drops to 19.5%. We set d to 128 for this experiment.

Weight α_4. Figure 3b shows that the weight α_4 also plays an important role. The best F_1 score is achieved when α_4 is set to 2 or 4 on this dataset.

Embedding Dimensionality. Figure 3c shows that increasing the embedding dimensionality in general yields better node embeddings. On the other hand, we notice that FastRP already achieves better performance than all baseline method with embedding dimensionality of 256.

4.8 Scalability

In Section 3.4, we show that the time complexity of FastRP is linear to the number of nodes n and the number of edges m. Here, we empirically verifies this by learning embeddings on random graphs generated by the Erdos-Renyi model. In Figure 4a, we fix m to 10^7 and vary n from 10^5 to 10^6. In Figure 4b, we fix n to 10^6 and vary m from 10^7 to 10^8. For both figures, we report the CPU time for FastRP to embed the graph. It can be seen that the empirical running time of FastRP also scales linearly with n and m.

5 RELATED WORK

Network embeddings. Most of the early methods view network embeddings as a dimension reduction problem, where the goal is to preserve the local or global distances between data points in a low-dimensional manifold [4, 27, 32]. With time complexity at least quadratic in the number of data points (or nodes), these methods do not scale to graphs with hundreds of thousands of nodes.

Inspired by the success of scalable neural methods for learning word embeddings, in particular the Skip-gram model [22], neural methods are proposed for network embeddings [16, 23, 30]. These methods typically sample node pairs that are close to each other and then train a Skip-gram model on the pairs to obtain node embeddings. The difference mostly lies in the strategy for node pairs sampling. DeepWalk [23] samples node pairs that are at most k hops away via random walking on the graph. Node2vec [16] introduces a biased random walk strategy using a mixture of DFS and BFS. LINE [30] considers the node pairs that are 1-hop or 2-hops away from each other. These methods not only produce high-quality node embeddings but also scale to networks with millions of nodes.

In Levy and Goldberg's seminal work [19] on interpreting Skip-gram with negative sampling (SGNS), they prove that SGNS implicitly factorizes a shifted pointwise mutual information (PMI) matrix of word co-occurrences. Using a similar methodology, it is shown that methods like DeepWalk [23], LINE [23], PTE [29] and node2vec [16] all implicitly approximate and factorize a node similarity matrix, which is usually some transformation of the k-step transition matrices $\mathbf{A}^k$ [26, 37] . Following these analyses, matrix

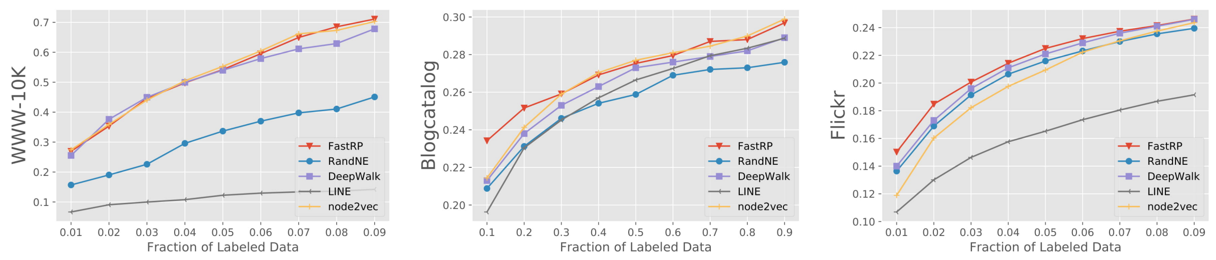
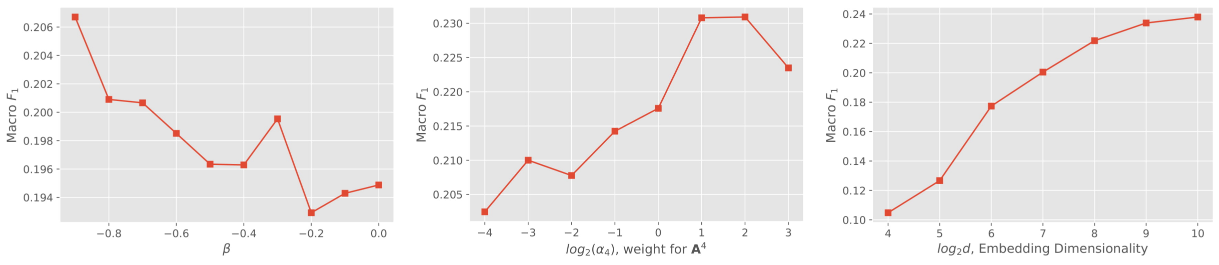
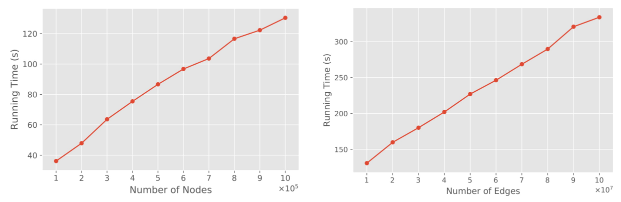


Figure 2: Detailed multi-label classification result on `WWW-10K`, `BlogCatalog`, and `Flickr` (Section 4.6).



(a) Normalization strength β. **(b) Weight α_4.** **(c) Embedding dimensionality d.**

Figure 3: Parameter sensitivity study on `Blogcatalog`.



(a) Change n. **(b) Change m.**

Figure 4: Scalability study on Erdos-Renyi graphs.

factorization-based methods are also proposed for network embeddings [7, 26, 37]. A representative method is GraRep [7], which can be seen as the matrix factorization version of DeepWalk: it uses SVD to factorize the shifted PMI matrix of k-step transition matrices. However, GraRep is not scalable due to the high time complexity of both raising the transition matrix A to higher powers and taking the element-wise logarithm of $\mathbf{A}^k$, which is a dense $n \times n$ matrix.

A few recent work thus propose to speed up the construction of such a node similarity matrix [25, 26, 39], which are inspired by the spectral graph theory. The basic idea is that if the top-h eigendecomposition of $\mathbf{A}$ is given by $\mathbf{A} = \mathbf{U}_h\Lambda_h\mathbf{U}_h^\top$, then $\mathbf{A}^k$ can be approximated with $\mathbf{U}_h\Lambda_h^k\mathbf{U}_h^\top$ [26, 39]. The major drawback of these methods is that they need to get rid of the element-wise normalization (such as taking logarithm) on A^k to achieve better scalability; this harms the quality of embeddings [8]. A recent method [25] proposes to sparsify $\mathbf{A}^k$ for better scalability. However, the similarity matrix is still dense even after the sparsification: for a graph with $n = 10^6$ nodes and $m = 10^7$ edges, the number of entries in the sparsified matrix can be as high as 1.4×10^{11} [25].

Perhaps the most relevant work is RandNE [38], which considers a Gaussian random projection approach for network embeddings. There are three key differences between our work and RandNE:

(1) We are the first to identify the two key factors for constructing the node similarity matrix: high-order proximity preservation and element normalization. Specifically, the importance of normalization is overlooked in many previous studies, including RandNE [26, 38, 39].
(2) Based on theoretical analysis, we derive a normalization algorithm that properly downweights the influence of high-degree nodes in the node similarity matrix. An additional advantage of our normalization approach is that it can be formalized as a simple matrix multiplication operation, which enables fast iterative computation when combined with random projection.
(3) We explore the usage of very sparse random projection for network embeddings, which is more efficient than traditional Gaussian random projection. As shown in the experiments, FastRP achieves substantially better performance on challenging downstream tasks while being faster.

Graph-based Recommendation Systems. Our work is also related to graph-based recommendation systems, which consider a special kind of graph: the bipartite graph between users and items. Typically, the goal is to generate the top-K items that a user will be most interested in.

Several early work in this field emphasis on the importance of high-order, transitive relationships between users and items [9, 10, 14, 24] for top-K recommendation. Fouss et al. [14, 24] present P^3, which directly uses the entries in the third power of the transition matrix to rank items. Although achieving competitive recommendation performance, it is observed that the ranking of items in P^3 is strongly influenced by the popularity of items [9, 10]: popular items tend to dominate the recommendation list for most users. To this end, P^3_α [10] and RP^3_β [9] are proposed as re-weighted versions of P^3. Their idea of downweighting popularity items is similar to the normalization strategy in this paper. However, these methods are proposed as heuristics specifically for bipartite graphs and the task of top-K recommendation, which do not generalize to other scenarios. Moreover, the power of the transition matrix is either computed exactly [10] or approximated by sampling a significant number of random walks [9], both of which are not scalable.

6 CONCLUSION

We present FastRP, a scalable algorithm for obtaining distributed representations of nodes in a graph. FastRP first constructs a node similarity matrix that captures high-order proximity between nodes and then normalizes the matrix entries based on the convergence properties of $\mathbf{A}^k$. Very sparse random projection is applied to this similarity matrix to obtain node embeddings. Experimental results show that FastRP achieves three orders of magnitudes speedup over state-of-the-art method DeepWalk while producing embeddings of comparable or even better quality.

7 ACKNOWLEDGEMENTS

This work was partially supported by NSF grants IIS-1546113 and IIS-1927227.

REFERENCES

[1] Dimitris Achlioptas. 2003. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of computer and System Sciences* 66, 4 (2003), 671–687.

[2] Rosa I Arriaga and Santosh Vempala. 1999. An algorithmic theory of learning: Robust concepts and random projection. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*. IEEE, 616–623.

[3] Albert-László Barabási and Eric Bonabeau. 2003. Scale-free networks. *Scientific american* 288, 5 (2003), 60–69.

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

[5] Robert D Blumofe, Christopher F Joerg, Bradley C Kuszmaul, Charles E Leiserson, Keith H Randall, and Yuli Zhou. 1996. Cilk: An efficient multithreaded runtime system. *Journal of parallel and distributed computing* 37, 1 (1996), 55–69.

[6] Paolo Boldi and Sebastiano Vigna. 2014. Axioms for centrality. *Internet Mathematics* 10, 3-4 (2014), 222–262.

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

[8] Siheng Chen, Sufeng Niu, Leman Akoglu, Jelena Kovačević, and Christos Faloutsos. 2017. Fast, warped graph embedding: Unifying framework and one-click algorithm. *arXiv preprint arXiv:1702.05764* (2017).

[9] Fabian Christoffel, Bibek Paudel, Chris Newell, and Abraham Bernstein. 2015. Blockbusters and wallflowers: Accurate, diverse, and scalable recommendations with random walks. In *Proceedings of the 9th ACM Conference on Recommender Systems*. ACM, 163–170.

[10] Colin Cooper, Sang Hyuk Lee, Tomasz Radzik, and Yiannis Siantos. 2014. Random walks in recommender systems: exact computation and simulations. In *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 811–816.

[11] Common Crawl. [n. d.]. Common Crawl's Web graph data. http://commoncrawl.org/2017/05/hostgraph-2017-feb-mar-apr-crawls/. Accessed: 2018-12-01.

[12] Sanjoy Dasgupta and Anupam Gupta. 1999. An elementary proof of the Johnson-Lindenstrauss lemma. *International Computer Science Institute, Technical Report* 22, 1 (1999), 1–5.

[13] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of machine learning research* 9, Aug (2008), 1871–1874.

[14] Francois Fouss, Alain Pirotte, and Marco Saerens. 2005. A novel way of computing similarities between nodes of a graph, with application to collaborative recommendation. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*. IEEE Computer Society, 550–556.

[15] Georg Frobenius, Ferdinand Georg Frobenius, Ferdinand Georg Frobenius, Ferdinand Georg Frobenius, and Germany Mathematician. 1912. Über Matrizen aus nicht negativen Elementen. (1912).

[16] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[17] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53, 2 (2011), 217–288.

[18] Megha Khosla, Avishek Anand, and Vinay Setty. 2019. A Comprehensive Comparison of Unsupervised Network Representation Learning Methods. *arXiv preprint arXiv:1903.07902* (2019).

[19] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*. 2177–2185.

[20] Ping Li, Trevor J Hastie, and Kenneth W Church. 2006. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 287–296.

[21] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[23] 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.

[24] Alain Pirotte, Jean-Michel Renders, Marco Saerens, et al. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge & Data Engineering* 3 (2007), 355–369.

[25] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, and Kuansan Wang. 2019. NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization. (2019).

[26] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 459–467.

[27] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290, 5500 (2000), 2323–2326.

[28] Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management* 24, 5 (1988), 513–523.

[29] Jian Tang, Meng Qu, and Qiaozhu Mei. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1165–1174.

[30] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. 1067–1077.

[31] Lei Tang and Huan Liu. 2009. Scalable learning of collective behavior based on sparse social dimensions. In *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 1107–1116.

[32] 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.

[33] 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 on World Wide Web*. 539–548.

[34] John W Tukey et al. 1957. On the comparative anatomy of transformations. *The Annals of Mathematical Statistics* 28, 3 (1957), 602–632.

[35] Santosh S Vempala. 2005. *The random projection method*. Vol. 65. American Mathematical Soc.

[36] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1225–1234.

[37] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. 2015. Network representation learning with rich text information. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[38] Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang, and Wenwu Zhu. 2018. Billion-scale Network Embedding with Iterative Random Projection. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 787–796.

[39] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-order proximity preserved network embedding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2778–2786.