NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization

Jiezhong Qiu[†]
qiujz16@mails.tsinghua.edu.cn
Tsinghua University

Yuxiao Dong
yuxdong@microsoft.com
Microsoft Research, Redmond

Hao Ma[*]
haom@fb.com
Facebook AI

Jian Li
lijian83@tsinghua.edu.cn
Tsinghua University

Chi Wang
wang.chi@microsoft.com
Microsoft Research, Redmond

Kuansan Wang
kuansanw@microsoft.com
Microsoft Research, Redmond

Jie Tang[†]
jietang@tsinghua.edu.cn
Tsinghua University

ABSTRACT

We study the problem of large-scale network embedding, which aims to learn latent representations for network mining applications. Previous research shows that 1) popular network embedding benchmarks, such as DeepWalk, are in essence implicitly factorizing a matrix with a closed form, and 2) the explicit factorization of such matrix generates more powerful embeddings than existing methods. However, directly constructing and factorizing this matrix—which is dense—is prohibitively expensive in terms of both time and space, making it not scalable for large networks.

In this work, we present the algorithm of large-scale network embedding as sparse matrix factorization (NetSMF). NetSMF leverages theories from spectral sparsification to efficiently sparsify the aforementioned dense matrix, enabling significantly improved efficiency in embedding learning. The sparsified matrix is spectrally close to the original dense one with a theoretically bounded approximation error, which helps maintain the representation power of the learned embeddings. We conduct experiments on networks of various scales and types. Results show that among both popular benchmarks and factorization based methods, NetSMF is the only method that achieves both high efficiency and effectiveness. We show that NetSMF requires only 24 hours to generate effective embeddings for a large-scale academic collaboration network with tens of millions of nodes, while it would cost DeepWalk months and is computationally infeasible for the dense matrix factorization solution. The source code of NetSMF is publicly available[1].

[*]Work performed while at Microsoft Research.
[†]Also with Beijing National Research Center for Information Science and Technology (BNRist).
[1]https://github.com/xptree/NetSMF

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313446

ACM Reference Format:
Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. 2019. NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization . In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313446

1 INTRODUCTION

Recent years have witnessed the emergence of network embedding, which offers a revolutionary paradigm for modeling graphs and networks [16]. The goal of network embedding is to automatically learn latent representations for objects in networks, such as vertices and edges. Significant lines of research have shown that the latent representations are capable of capturing the structural properties of networks, facilitating various downstream network applications, such as vertex classification and link prediction [12, 14, 27, 33].

Over the course of its development, the DeepWalk [27], LINE [33], and node2vec [14] models have been commonly considered as powerful benchmark solutions for evaluating network embedding research. The advantage of LINE lies in its scalability for large-scale networks as it only models the first- and second-order proximities. That is to say, its embeddings lose the multi-hop dependencies in networks. DeepWalk and node2vec, on the other hand, leverage random walks on graphs and skip-gram [24] with large context sizes to model nodes further away (i.e., global structures). Consequently, it is computationally more expensive for DeepWalk and node2vec to handle large-scale networks. For example, with the default parameter settings [27], DeepWalk requires months to embed an academic collaboration network of 67 million vertices and 895 million edges[2]. The node2vec model, which performs high-order random walks, takes more time than DeepWalk to learn embeddings.

More recently, a study shows that both the DeepWalk and LINE methods can be viewed as implicit factorization of a closed-form matrix [28]. Building upon this theoretical foundation, the NetMF method was instead proposed to explicitly factorize this matrix,

[2]With the default DeepWalk parameters (walk length: 40 and #walk per node: 80), 214+ billion nodes (67M×40×80) with a vocabulary size of 67 million are fed into skip-gram. As a reference, Mikolov et al. reported that training on Google News of 6 billion words and a vocabulary size of only 1 million cost 2.5 days with 125 CPU cores [24].

Table 1: The comparison between NetSMF and other popular network embedding algorithms.

	LINE	DeepWalk	node2vec	NetMF	NetSMF
Efficiency	√				√
Global context		√	√	√	√
Theoretical guarantee				√	√
High-order proximity			√		

achieving more effective embeddings than DeepWalk and LINE. Unfortunately, it turns out that the matrix to be factorized is an $n \times n$ dense one with n being the number of vertices in the network, making it prohibitively expensive to directly construct and factorize for large-scale networks.

In light of these limitations of existing methods (See the summary in Table 1), we propose to study representation learning for large-scale networks with the goal of achieving efficiency, capturing global structural contexts, and having theoretical guarantees. Our idea is to find a sparse matrix that is spectrally close to the dense NetMF matrix implicitly factorized by DeepWalk. The sparsified matrix requires a lower cost for both construction and factorization. Meanwhile, making it spectrally close to the original NetMF matrix can guarantee that the spectral information of the network is maintained, and the embeddings learned from the sparse matrix is as powerful as those learned from the dense NetMF matrix.

In this work, we present the solution to network embedding learning as sparse matrix factorization (NetSMF). NetSMF comprises three steps. First, it leverages the spectral graph sparsification technique [7, 8] to find a sparsifier for a network's random-walk matrix-polynomial. Second, it uses this sparsifier to construct a matrix with significantly fewer non-zeros than, but spectrally close to, the original NetMF matrix. Finally, it performs randomized singular value decomposition to efficiently factorize the sparsified NetSMF matrix, yielding the embeddings for the network.

With this design, NetSMF offers both efficiency and effectiveness with guarantees, as the approximation error of the sparsified matrix is theoretically bounded. We conduct experiments in five networks, which are representative of different scales and types. Experimental results show that for million-scale or larger networks, NetSMF achieves orders of magnitude speedup over NetMF, while maintaining competitive performance for the vertex classification task. In other words, both NetSMF and NetMF outperform well-recognized network embedding benchmarks (i.e., DeepWalk, LINE, and node2vec), but NetSMF addresses the computation challenge faced by NetMF.

To summarize, we introduce the idea of network embedding as sparse matrix factorization and present the NetSMF algorithm, which makes the following contributions to network embedding:

Efficiency. NetSMF reaches significantly lower time and space complexity than NetMF. Remarkably, NetSMF is able to generate embeddings for a large-scale academic network of 67 million vertices and 895 million edges on a single server in 24 hours, while it would cost months for DeepWalk and node2vec, and is computationally infeasible for NetMF on the same hardware.

Table 2: Notations.

Notation	Description
G	input network
V	vertex set of G with $\|V\|$=n
E	edge set of G with $\|E\| = m$
$\boldsymbol{A}$	adjacency matrix of G
$\boldsymbol{D}$	degree matrix of G
$\mathrm{vol}(G)$	volume of G
b	number of negative samples
T	context window size
d	embedding dimension
$\boldsymbol{L}$	random-walk molynomial of G (Eq. (4))
$\widehat{\boldsymbol{L}}$	$\boldsymbol{L}$'s sparsifier
$\boldsymbol{M}$	$\frac{1}{T}\sum_{r=1}^{T}(\boldsymbol{D}^{-1}\boldsymbol{A})^r\boldsymbol{D}^{-1}$
$\widehat{\boldsymbol{M}}$	$\boldsymbol{M}$'s sparsifier
$\mathrm{trunc\_log}^{\circ}\ \frac{\mathrm{vol}(G)}{b}\boldsymbol{M}$	NetMF matrix
$\mathrm{trunc\_log}^{\circ}\ \frac{\mathrm{vol}(G)}{b}\widehat{\boldsymbol{M}}$	NetMF matrix sparisifier
M	number of non-zeros in $\widehat{\boldsymbol{L}}$
ϵ	approximation factor
$[x]$	set $\{1; 2; \cdots; x\}$ for positive integer x

Effectiveness. NetSMF is capable of learning embeddings that maintain the same representation power as the dense matrix factorization solution, making it consistently outperform DeepWalk and node2vec by up to 34% and LINE by up to 100% for the multi-label vertex classification task in networks.

Theoretical Guarantee. NetSMF's efficiency and effectiveness are theoretically backed up. The sparse NetSMF matrix is spectrally close to the exact NetMF matrix, and the approximation error can be bounded, maintaining the representation power of its sparsely learned embeddings.

2 PRELIMINARIES

Commonly, the problem of network embedding is formalized as follows: Given an undirected and weighted network $G = (V, E, A)$ with V as the vertex set of n vertices, E as the edge set of m edges, and A as the adjacency matrix, the goal is to learn a function $V \to \mathrm{R}^d$ that maps each vertex to a d-dimensional ($d \ll n$) vector that captures its structural properties, e.g., community structures. The vector representation of each vertex can be fed into downstream applications such as link prediction and vertex classification.

One of the pioneering work on network embedding is the DeepWalk model [27], which has been consistently considered as a powerful benchmark over the past years [16]. In brief, DeepWalk is coupled with two steps. First, it generates several vertex sequences by random walks over a network; Second, it applies the skip-gram model [25] on the generated vertex sequences to learn the latent representations for each vertex. Commonly, skip-gram is parameterized with the context window size T and the number of negative samples b. Recently, a theoretical study [28] reveals that DeepWalk essentially factorizes a matrix derived from the random walk process. More formally, it proves that when the length of random walks goes to infinity, DeepWalk implicitly and asymptotically factorizes

the following matrix:

$$\log^{\circ}\left(\frac{\mathrm{vol}(G)}{b}\boldsymbol{M}\right); \tag{1}$$

where $\mathrm{vol}(G) = \sum_i \sum_j A_{ij}$ denotes the volume of the graph, and

$$\boldsymbol{M} = \frac{1}{T}\sum_{r=1}^{T}(\boldsymbol{D}^{-1}\boldsymbol{A})^r\boldsymbol{D}^{-1}; \tag{2}$$

where $\boldsymbol{D} = \mathrm{diag}(d_1, \cdots, d_n)$ is the degree matrix with $d_i = \sum_j A_{ij}$ as the generalized degree of the i-th vertex. Note that $\log^{\circ}(\,)$ represents the element-wise matrix logarithm [18], which is different from the matrix logarithm. In other words, the matrix in Eq. (1) can be characterized as the result of applying element-wise matrix logarithm (i.e., $\log^{\circ}$) to matrix $\frac{\mathrm{vol}(G)}{b}\boldsymbol{M}$.

The matrix in Eq. (1) offers an alternative view of the skip-gram based network embedding methods. Further, Qiu et al. provide an explicit matrix factorization approach named NetMF to learn the embeddings [28]. It shows that the accuracy for vertex classification based on the embeddings from NetMF outperforms that based on DeepWalk and LINE. Note that the matrix in Eq. (1) would be ill-defined if there exist a pair of vertices unreachable in T hops, because $\log(0) = -\infty$. So following Levy and Goldberg [22], NetMF uses the logarithm truncated at point one, that is, $\mathrm{trunc\_log}(x) = \max(0, \log(x))$. Thus, NetMF targets to factorize the matrix

$$\mathrm{trunc\_log}^{\circ}\left(\frac{\mathrm{vol}(G)}{b}\boldsymbol{M}\right). \tag{3}$$

In the rest of this work, we refer to the matrix in Eq. (3) as *the NetMF matrix*.

However, there exist a couple of challenges when leveraging the NetMF matrix in practice. First, almost every pair of vertices within distance $r \le T$ correspond to a non-zero entry in the NetMF matrix. Recall that many social and information networks exhibit the small-world property where most vertices can be reached from each other in a small number of steps. For example, as of the year 2012, 92% of the reachable pairs in Facebook are at distance five or less [2]. As a consequence, even if setting a moderate context window size (e.g., the default setting $T = 10$ in DeepWalk), the NetMF matrix in Eq. (3) would be a dense matrix with $O(n^2)$ number of non-zeros. The exact construction and factorization of such a matrix is impractical for large-scale networks. More concretely, computing the matrix power in Eq. (2) involves dense matrix multiplication which costs $O(n^3)$ time; factorizing a $n \times n$ dense matrix is also time consuming. To reduce the construction cost, NetMF approximates $\boldsymbol{M}$ with its top eigen pairs. However, the approximated matrix is still dense, making this strategy unable to handle large networks.

In this work, we aim to address the efficiency and scalability limitation of NetMF, while maintaining its superiority in effectiveness. We list necessary notations and their descriptions in Table 2.

3 NETWORK EMBEDDING AS SPARSE MATRIX FACTORIZATION (NetSMF)

In this section, we develop network embedding as sparse matrix factorization (NetSMF). We present the NetSMF method to construct and factorize a sparse matrix that approximates the dense NetMF matrix. The main technique we leverage is random-walk matrix-polynomial (molynomial) sparsification.

3.1 Random-Walk Molynomial Sparsification

We first introduce the definition of spectral similarity and the theorem of random-walk molynomial sparsification.

Definition 1. *(Spectral Similarity of Networks) Suppose $G = (V, E, A)$ and $\mathcal{G} = (V, \mathcal{E}, \mathcal{A})$ are two weighted undirected networks. Let $\boldsymbol{L} = \boldsymbol{D}_G - \boldsymbol{A}$ and $\mathcal{L} = \boldsymbol{D}_{\mathcal{G}} - \mathcal{A}$ be their Laplacian matrices, respectively. We define G and $\mathcal{G}$ are $(1+\epsilon)$-spectrally similar if*

$$\forall \boldsymbol{x} \in \mathbb{R}^n, (1-\epsilon)\cdot \boldsymbol{x}^\top \mathcal{L}\boldsymbol{x} \le \boldsymbol{x}^\top \boldsymbol{L}\boldsymbol{x} \le (1+\epsilon)\cdot \boldsymbol{x}^\top \mathcal{L}\boldsymbol{x}.$$

Theorem 1. *(Spectral Sparsifiers of Random-Walk Molynomials [7, 8]) For random-walk molynomial*

$$\boldsymbol{L} = \boldsymbol{D} - \sum_{r=1}^{T}\alpha_r \boldsymbol{D}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^r; \tag{4}$$

where $\sum_{r=1}^{T}\alpha_r = 1$ and α_r non-negative, one can construct, in time $O(T^2 m\epsilon^{-2}\log^2 n)$, a $(1+\epsilon)$-spectral sparsifier, $\mathcal{L}$, with $O(n\log n\epsilon^{-2})$ non-zeros. For unweighted graphs, the time complexity can be reduced to $O(T^2 m\epsilon^{-2}\log n)$.

To achieve a sparsifier $\mathcal{L}$ with $O(\epsilon^{-2}n\log n)$ non-zeros, the sparsification algorithm consists of two steps: The first step obtains an initial sparsifier for $\boldsymbol{L}$ with $O(Tm\epsilon^{-2}\log n)$ non-zeros. The second step then applies the standard spectral sparsification algorithm [30] to further reduce the number of non-zeros to $O(\epsilon^{-2}n\log n)$. In this work, we only adopt the first step because a sparsifier with $O(Tm\epsilon^{-2}\log n)$ non-zeros is sparse enough for our task. Thus we skip the second step that involves additional computations. From now on, when referring to the random-walk molynomial sparsification algorithm in this work, we mean its first step only.

One can immediately observe that, if we set $\alpha_r = \frac{1}{T}, r \in [T]$, the matrix $\boldsymbol{L}$ in Eq. (4) has a strong connection with the desired matrix $\boldsymbol{M}$ in Eq. (2). Formally, we have the following equation

$$\boldsymbol{M} = \boldsymbol{D}^{-1}(\boldsymbol{D} - \boldsymbol{L})\boldsymbol{D}^{-1}. \tag{5}$$

Thm. 1 can help us construct a sparsifier $\mathcal{L}$ for matrix $\boldsymbol{L}$. Then we define $\mathcal{M} \triangleq \boldsymbol{D}^{-1}(\boldsymbol{D} - \mathcal{L})\boldsymbol{D}^{-1}$ by replacing $\boldsymbol{L}$ in Eq. (5) with its sparsifier $\mathcal{L}$. One can observe that matrix $\mathcal{M}$ is still a sparse one with the same order of magnitude of non-zeros as $\mathcal{L}$. Consequently, instead of factorizing the dense NetMF matrix in Eq. (3), we can factorize its sparse alternative, i.e.,

$$\mathrm{trunc\_log}^{\circ}\left(\frac{\mathrm{vol}(G)}{b}\mathcal{M}\right). \tag{6}$$

In the rest of this work, the matrix in Eq. (6) is referred to as *the NetMF matrix sparsifier*.

3.2 The NetSMF Algorithm

In this section, we formally describe the NetSMF algorithm, which consists of three steps: random-walk molynomial sparsification, NetMF sparsifier construction, and truncated singular value decomposition.

Step 1: Random-Walk Molynomial Sparsification. To achieve the sparsifier $\mathcal{L}$, we adopt the algorithm in Cheng et al. [8]. The algorithm starts from creating a network $\mathcal{G}$ that has the same vertex set as G and an empty edge set (Alg. 1, Line 1). Next, the algorithm constructs a sparsifier with $O(M)$ non-zeros by repeating the PathSampling algorithm for M times. In each iteration, it picks an edge

Algorithm 1: NetSMF

Input : A social network $G = (V, E, \boldsymbol{A})$ which we want to learn network embedding; The number of non-zeros M in the sparsifier; The dimension of embedding d.

Output : An embedding matrix of size $n \times d$, each row corresponding to a vertex.

1 $\mathcal{G} \leftarrow (V, \emptyset, \mathcal{A} = \mathbf{0})$

/* Create an empty network with $E = \emptyset$ and $\mathcal{A} = 0$. */

2 **for** $i \leftarrow 1$ **to** M **do**

3 Uniformly pick an edge $e = (u, v) \in E$

4 Uniformly pick an integer $r \in [T]$

5 $u', v'; Z \leftarrow$ PathSampling(e, r)

6 Add an edge $u', v'; \frac{2rm}{MZ}$ to $\mathcal{G}$

/* Parallel edges will be merged into one edge, with their weights summed up together. */

7 **end**

8 Compute $\mathcal{L}$ to be the unnormalized graph Laplacian of $\mathcal{G}$

9 Compute $\mathcal{M} = D^{-1}\left(D - \mathcal{L}\right)D^{-1}$

10 $U_d, \Sigma_d, V_d \leftarrow$ RandomizedSVD(trunc_log$^{\circ}\left(\frac{\mathrm{vol}(G)}{b}\mathcal{M}\right), d$)

11 **return** $U_d\sqrt{\Sigma_d}$ as network embeddings

Algorithm 2: PathSampling algorithm as described in [8].

1 **Procedure** PathSampling($e = (u, v), r$)

2 Uniformly pick an integer $k \in [r]$

3 Perform $(k-1)$-step random walk from u to u_0

4 Perform $(r-k)$-step random walk from v to u_r

5 Keep track of $Z(\boldsymbol{p})$ along the length-r path $\boldsymbol{p}$ between u_0 and u_r according to Eq. (7)

6 **return** $u_0, u_r, Z(\boldsymbol{p})$

$e \in E$ and an integer $r \in [T]$ uniformly (Alg. 1, Line 3-4). Then, the algorithm uniformly draws an integer $k \in [r]$ and performs $(k-1)$-step and $(r-k)$-step random walks starting from the two endpoints of edge e respectively (Alg. 2, Line 3-4). The above process samples a length-r path $\boldsymbol{p} = (u_0, u_1, \cdots, u_r)$. At the same time, the algorithm keeps track of $Z(\boldsymbol{p})$, which is defined by

$$Z(\boldsymbol{p}) = \sum_{i=1}^{r} \frac{2}{\boldsymbol{A}_{u_{i-1}, u_i}}; \tag{7}$$

and then adds a new edge (u_0, u_r) with weight $\frac{2rm}{MZ(\boldsymbol{p})}$ to $\mathcal{G}$ (Alg. 1, Line 6).[3] Parallel edges in $\mathcal{G}$ will be merged into one single edge, with their weights summed up together. Finally, the algorithm computes the Laplacian of $\mathcal{G}$, which is the sparsifier $\mathcal{L}$ as we desired (Alg. 1, Line 8). This step gives us a sparsifier with $O(M)$ non-zeros.

Step 2: Construct a NetMF Matrix Sparsifier. As we have discussed at the end of Section 3.1, after constructing a sparsifier $\mathcal{L}$, we can plug it into Eq. (5) to obtain a NetMF matrix sparsifier as shown in Eq. (6) (Alg. 1, Line 9-10). This step does not change the order of magnitude of non-zeros in the sparsifier.

[3] Details about how the edge weight is derived can be found in Thm. 4 in Appendix.

Algorithm 3: Randomized SVD on NetMF Matrix Sparsifier

/* In this work, the matrix to be factorized (Eq. (6)) is an $n \times n$ symmetric sparse matrix. We store this sparse matrix in a row-major way and make use of its symmetry to simplify the computation. */

1 **Procedure** RandomizedSVD($\boldsymbol{A}, d$)

2 Sampling Gaussian random matrix $\boldsymbol{O}$ // $\boldsymbol{O} \in \mathbb{R}^{n \times d}$

3 Compute sample matrix $Y = \boldsymbol{A}^{\top}\boldsymbol{O} = \boldsymbol{A}\boldsymbol{O}$ // $Y \in \mathbb{R}^{n \times d}$

4 Orthonormalize Y

5 Compute $\boldsymbol{B} = \boldsymbol{A}Y$ // $\boldsymbol{B} \in \mathbb{R}^{n \times d}$

6 Sample another Gaussian random matrix $\boldsymbol{P}$ // $\boldsymbol{P} \in \mathbb{R}^{d \times d}$

7 Compute sample matrix of $Z = \boldsymbol{B}\boldsymbol{P}$ // $Z \in \mathbb{R}^{n \times d}$

8 Orthonormalize Z

9 Compute $\boldsymbol{C} = Z^{\top}\boldsymbol{B}$ // $\boldsymbol{C} \in \mathbb{R}^{d \times d}$

10 Run Jacobi SVD on $\boldsymbol{C} = \boldsymbol{U}\Sigma\boldsymbol{V}^{\top}$

11 **return** $Z\boldsymbol{U}, \Sigma, Y\boldsymbol{V}$

/* Result matrices are of shape $n \times d$; $d \times d$; $n \times d$ resp. */

Table 3: Time and Space Complexity of NetSMF.

	Time	Space
Step 1	$O(MT \log n)$ for weighted networks $O(MT)$ for unweighted networks	$O(M + n + m)$
Step 2	$O(M)$	$O(M + n)$
Step 3	$O(Md + nd^2 + d^3)$	$O(M + nd)$

Step 3: Truncated Singular Value Decomposition. The final step is to perform truncated singular value decomposition (SVD) on the constructed NetMF matrix sparsifier (Eq. (6)). However, even the sparsifier only has $O(M)$ number of non-zeros, performing exact SVD is still time consuming. In this work, we leverage a modern randomized matrix approximation technique—Randomized SVD—developed by Halko et al. [15]. Due to space constraint, we cannot include many details. Briefly speaking, the algorithm projects the original matrix to a low-dimensional space through a Gaussian random matrix. One only needs to perform traditional SVD (e.g. Jacobi SVD) on a $d \times d$ small matrix. We list the pseudocode algorithm in Alg. 3. Another advantage of SVD is that we can determine the dimensionality of embeddings by using, for example, Cattell's Scree test [5]. In the test, we plot the singular values and select a rank d such that there is a clear drop in the magnitudes or the singular values start to even out. More details will be discussed in Section 4.

Complexity Analysis. Now we analyze the time and space complexity of NetSMF, as summarized in Table 3. As for step 1, we call the PathSampling algorithm for M times, during each of which it performs $O(T)$ steps of random walks over the network. For unweighted networks, sampling a neighbor requires $O(1)$ time, while for weighted networks, one can use roulette wheel selection to choose a neighbor in $O(\log n)$. It taks $O(M)$ space to store $\mathcal{G}$, while the additional $O(n + m)$ space comes from the storage of the input network. As for step 2, it takes $O(M)$ time to perform the transformation in Eq. (5) and the element-wise truncated logarithm in Eq. (6). The additional $O(n)$ space is spent in storing the degree matrix. As for step 3, $O(Md)$ time is required to compute the product of a row-major sparse matrix and a dense matrix (Alg. 3, Lines 3 and

5); $O(nd^2)$ time is spent in Gram-Schmidt orthogonalization (Alg. 3, Lines 4 and 8); $O(d^3)$ time is spent in Jacobi SVD (Alg. 3, Line 10).

Connection to NetMF. The major difference between NetMF and NetSMF lies in the approximation strategy of the NetMF matrix in Eq. (3). As we mentioned in Section 2, NetMF approximates it with a dense matrix, which brings new space and computation challenges. In this work, NetSMF aims to find a sparse approximator to the NetMF matrix by leveraging theories and techniques from spectral graph sparsification.

Example. We provide a running example to help understand the NetSMF algorithm. Suppose we want to learn embeddings for a network with $n = 10^6$ vertices, $m = 10^7$ edges, context window size $T = 10$, and approximation factor $\epsilon = 0.1$. The NetSMF method calls the PathSampling algorithm for $M = Tm\epsilon^{-2}\log n \approx 1.4\times10^{11}$ times and provides us with a NetMF matrix sparsifier with at most 1.4×10^{11} non-zeros (Notice that the reducer in Step 1 and trunc_log° in Step 2 will further sparsify the matrix, making 1.4×10^{11} an upper bound). The density of the sparsifier is at most $\frac{M}{n^2} \approx 14\%$. Then, when computing the sparse-dense matrix product in randomized SVD (Alg. 3, Lines 3 and 5), the sparseness of the factorized matrix can greatly accelerate the calculation. In comparison, NetMF must construct a dense matrix with $n^2 = 10^{12}$ non-zeros, which is an order of magnitude larger in terms of density. Also, the density of the sparsifier in NetSMF can be further reduced by using a larger ϵ, while NetMF does not have this flexibility.

3.3 Approximation Error Analysis

In this section, we analyze the approximation error of the sparsification. We assume that we choose an approximation factor $\epsilon < 0.5$. We first see how the constructed $\widetilde{M}$ approximates M and then compare the NetMF matrix (Eq. (3)) against the NetMF matrix sparsifier (Eq. (6)). We use σ_i to denote the i-th descending-order singular value of a matrix. We also assume the vertices' degrees are sorted in ascending order, that is, $d_{\min} = d_1 \le d_2 \cdots \le d_n$.

Theorem 2. *The singular value of $\widetilde{M} - M$ satisfies $\sigma_i(\widetilde{M} - M) \le \frac{4\epsilon}{\sqrt{d_i d_{\min}}}, \forall i \in [n]$.*

Theorem 3. *Let $\|\cdot\|_F$ be the matrix Frobenius norm. Then*

$$\left\| \text{trunc\_log}^{\circ}\left(\frac{\text{vol}(G)}{b}\widetilde{M}\right) - \text{trunc\_log}^{\circ}\left(\frac{\text{vol}(G)}{b}M\right) \right\|_F \le \frac{4\epsilon\,\text{vol}(G)}{b\sqrt{d_{\min}}}\sqrt{\sum_{i=1}^{n}\frac{1}{d_i}}.$$

Proof. See Appendix.

Discussion on the Approximation Error. The above bound is achieved without making assumptions about the input network. If we introduce some assumptions, say a bounded lowest degree $d_{\min}$ or a specific random graph model (e.g., Planted Partition Model or Extended Planted Partition Model), it is promising to explore tighter bounds by leveraging theorems in literature [6, 11].

3.4 Parallelization

Each step of NetSMF can be parallelized, enabling it to scale to very large networks. The parallelization design of NetSMF is introduced in Figure 1. Below we discuss the parallelization of each step in detail. At the first step, the paths in the PathSampling algorithm are sampled independently with each other. Thus we can launch multiple PathSampling workers simultaneously. Each worker handles a subset of the samples. Herein, we require that each worker is able to access the network data $G = (V, E, A)$ efficiently. There are many options to meet this requirement. The easiest one is to load a copy of the network data to each worker's memory. When the network is extremely large (e.g., trillion scale) or workers have memory constraints, the graph engine should be designed to expose efficient graph query APIs to support graph operations such as random walks. At the end of this step, a reducer is designed to merge parallel edges and sum up their weights. If this step is implemented in a big data system such as Spark [45], the reduction step can be simply achieved by running a reduceByKey(_+_)[4] function. After the reduction, the sparsifier $\widetilde{E}$ is organized as a collection of triplets, a.k.a, COOrdinate format, with each indicating an entry of the sparsifier. The second step is the most straightforward step to scale up. When processing a triplet (u, v, w), we can simply query the degree of vertices u and v and perform the transformation defined in Eq. (5) as well as the truncated logarithm in Eq. (6), which can be well parallelized. For the last step, we organize the sparsifier into row-major format. This format allows efficient multiplication between a sparse and a dense matrix (Alg. 3, Line 3 and 5). Other dense matrix operators (e.g., Gaussian random matrix generation, Gram-Schmidt orthogonalization and Jacobi SVD) can be easily accelerated by using multi-threading or common linear algebra libraries. In this work, we adopt a single-machine shared-memory implementation. We use OpenMP [10] to parallelize NetSMF in our implementation[5].

Table 4: Statistics of Datasets.

Dataset	BlogCatalog	PPI	Flickr	YouTube	OAG
$\|V\|$	10,312	3,890	80,513	1,138,499	67,768,244
$\|E\|$	333,983	76,584	5,899,882	2,990,443	895,368,962
#labels	39	50	195	47	19

4 EXPERIMENTS

In this section, we evaluate the proposed NetSMF method on the multi-label vertex classification task, which has been commonly used to evaluate previous network embedding techniques [14, 27, 28, 33]. We introduce our datasets and baselines in Section 4.1 and Section 4.2. We report experimental results and parameter analysis in Section 4.3 and Section 4.4, respectively.

4.1 Datasets

We employ five datasets for the prediction task, four of which are in relatively small scale but have been widely used in network embedding literature, including BlogCatalog, PPI, Flickr, and YouTube. The remaining one is a large-scale academic co-authorship network, which is at least two orders of magnitude larger than the largest one (YouTube) used in most network embedding studies. The statistics of these datasets are listed in Table 4.

[4] https://spark.apache.org/docs/latest/rdd-programming-guide.html
[5] Code is publicly available at https://github.com/xptree/NetSMF

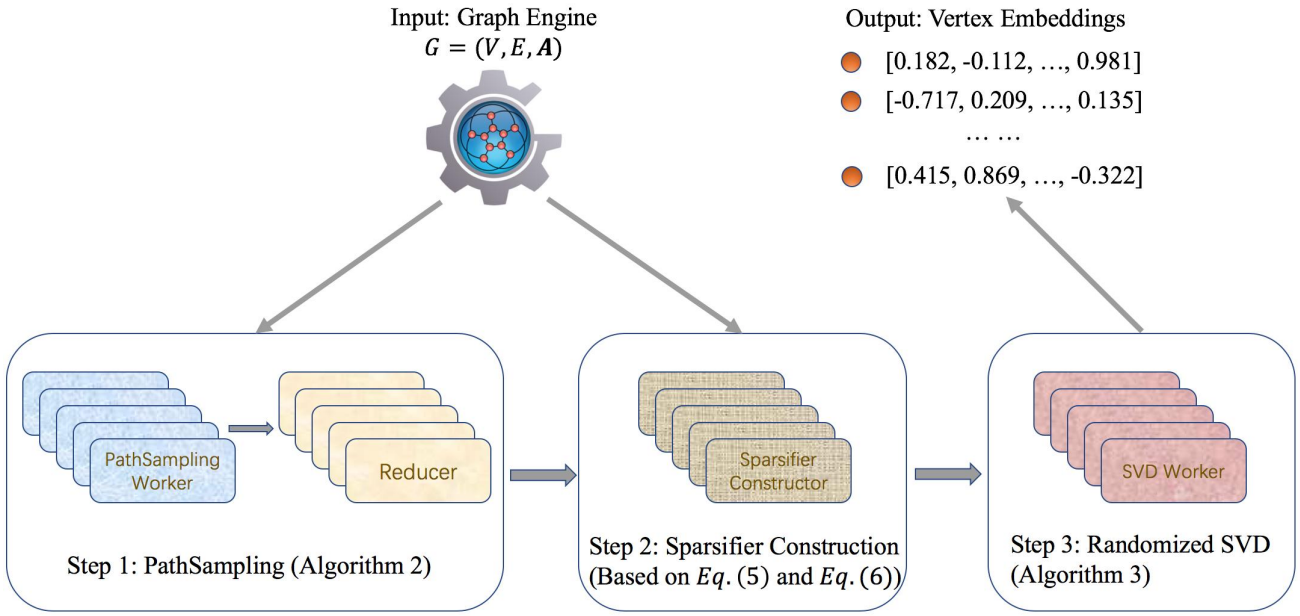


Figure 1: The System Design of NetSMF. The input comes from a graph engine which stores the network data and provides efficient APIs to graph queries. In Step 1, the system launches several PathSampling workers. Each worker handles a subset of samples. Then, a reducer is designed to aggregate the output of the PathSampling algorithm. In Step 2, the system distributes data to several sparsifier constructors to perform the transformation defined in Eq. (5) and the truncated element-wise matrix logarithm in Eq. (6). In the final step, the system applies truncated randomized SVD on the constructed sparsifier and dumps the resulted embeddings to storage.

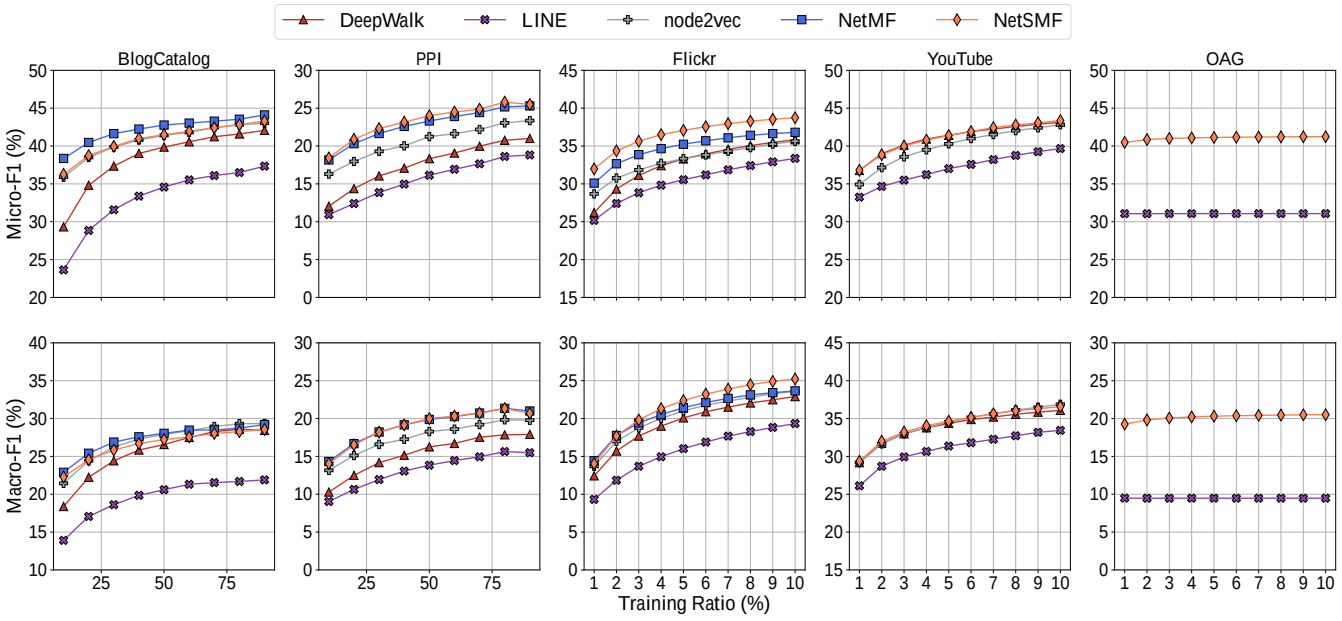


Figure 2: Predictive performance w.r.t. the ratio of training data. The x-axis represents the ratio of labeled data (%), and the y-axis in the top and bottom rows denote the Micro-F1 and Macro-F1 scores (%) respectively. For methods which fail to finish computation in one week or cannot handle the computation, their results are not available and thus not plotted in this figure.

BlogCatalog [1, 35] is a network of social relationships of online bloggers. The vertex labels represent the interests of the bloggers.

Protein-Protein Interactions (PPI) [31] is a subgraph of the PPI network for Homo Sapiens. The vertex labels are obtained from the hallmark gene sets and represent biological states.

Flickr [35] is the user contact network in Flickr. The labels represent the interest groups of the users.

YouTube [36] is a video-sharing website that allows users to upload, view, rate, share, add to their favorites, report, comment on videos. The users are labeled by the video genres they liked.

Open Academic Graph (OAG)[6] is an academic graph indexed by Microsoft Academic [29] and AMiner.org [34]. We construct an undirected co-authorship network from OAG, which contains 67,768,244 authors and 895,368,962 collaboration edges. The vertex labels are defined to be the top-level fields of study of each author, such as computer science, physics and psychology. In total, there are 19 distinct fields (labels) and authors may publish in more than one field, making the associated vertices have multiple labels.

4.2 Baseline Methods

We compare NetSMF with NetMF [28], LINE [33], DeepWalk [27], and node2vec [14]. For NetSMF, NetMF, DeepWalk, and node2vec that allow multi-hop structural dependencies, the context window size T is set to be 10, which is also the default setting used in both DeepWalk and node2vec. Across all datasets, we set the embedding dimension d to be 128. We follow the common practice for the other hyper-parameter settings, which are introduced below.

LINE. We use LINE with the second order proximity (i.e., LINE (2nd) [33]). We use the default setting of LINE's hyper-parameters: the number of edge samples to be 10 billion and the negative sample size to be 5.

DeepWalk. We present DeepWalk's results with the authors' preferred parameters, that is, walk length to be 40, the number of walks from each vertex to be 80, and the number of negative samples in skip-gram to be 5.

node2vec. For the return parameter p and in-out parameter q in node2vec, we adopt the default setting that was used by its authors if available. Otherwise, we grid search $p, q \in \{0.25, 0.5, 1, 2, 4\}$. For a fair comparison, we use the same walk length and the number of walks per vertex as DeepWalk.

NetMF. In NetMF, the hyper-parameter h indicates the number of eigen pairs used to approximate the NetMF matrix. We choose $h = 256$ for the BlogCatalog, PPI and Flickr datasets.

NetSMF. In NetSMF, we set the number of samples $M = 10^3 \times T \times m$ for the PPI, Flickr, and YouTube datasets, $M = 10^4 \times T \times m$ for BlogCatalog, and $M = 10 \times T \times m$ for OAG in order to achieve desired performance. For both NetMF and NetSMF, we have $b = 1$.

Prediction Setting. We follow the same experiment and evaluation procedures that were performed in DeepWalk [27]. First, we randomly sample a portion of labeled vertices for training and use the remaining for testing. For the BlogCatalog and PPI datasets, the training ratio varies from 10% to 90%. For Flickr, YouTube and OAG, the training ratio varies from 1% to 10%. We use the one-vs-rest logistic regression model implemented by LIBLINEAR [13] for the multi-label vertex classification task. In the test phase, the one-vs-rest model yields a ranking of labels rather than an exact label assignment. To avoid the thresholding effect, we take the assumption that was made in DeepWalk, LINE, and node2vec, that is, the number of labels for vertices in the test data is given [14, 27, 37]. We repeat the prediction procedure ten times and evaluate the average performance in terms of both Micro-F1 and Macro-F1 scores [41]. All the experiments are performed on a server with Intel Xeon E7-8890 CPU (64 cores), 1.7TB memory, and 2TB SSD hard drive.

[6] www.openacademic.ai/oag/

Table 5: Efficiency comparison. The running time includes filesystem IO and computation time. "–" indicates that the corresponding algorithm fails to complete within one week. "×" indicates that the corresponding algorithm is unable to handle the computation due to excessive space and memory consumption.

	LINE	DeepWalk	node2vec	NetMF	NetSMF
BlogCatalog	40 mins	12 mins	56 mins	2 mins	13 mins
PPI	41 mins	4 mins	4 mins	16 secs	10 secs
Flickr	42 mins	2.2 hours	21 hours	2 hours	48 mins
YouTube	46 mins	1 day	4 days	×	4.1 hours
OAG	2.6 hours	–	–	×	24 hours

4.3 Experimental Results

We summarize the prediction performance in Figure 2. To compare the efficiency of different algorithms, we also list the running time of each algorithm across all datasets, if available, in Table 5.

NetSMF vs. NetMF. We first focus on the comparison between NetSMF and NetMF, since the goal of NetSMF is to address the efficiency and scalability issues of NetMF while maintaining its superiority in effectiveness. From Table 5, we observe that for YouTube and OAG, both of which contain more than one million vertices, NetMF fails to complete because of the excessive space and memory consumption, while NetSMF is able to finish in four hours and one day, respectively. For the moderate-size network Flickr, both methods are able to complete within one week, though NetSMF is 2.5× faster (i.e., 48 mins vs. 2 hours). For small-scale networks, NetMF is faster than NetSMF in BlogCatalog and is comparable to NetSMF in PPI in terms of running time. This is because when the input networks contain only thousands of vertices, the advantage of sparse matrix construction and factorization over its dense alternative could be marginalized by other components of the workflow.

In terms of prediction performance, Figure 2 suggests NetSMF and NetMF yield consistently the best results among all compared methods, empirically demonstrating the power of the matrix factorization framework for network embedding. In BlogCatalog, NetSMF has slightly worse performance than NetMF (on average less than 3.1% worse regarding both Micro- and Macro-F1). In PPI, the two leading methods' performance are relatively indistinguishable in

terms of both metrics. In Flickr, NetSMF achieves significantly better Macro-F1 than NetMF (by 3.6% on average), and also higher Micro-F1 (by 5.3% on average). Recall that NetMF uses a dense approximation of the matrix to factorize. These results show that the sparse spectral approximation used by NetSMF does not necessarily yield worse performance than the dense approximation used by NetMF.

Overall, not only NetSMF improves the scalability, and the running time of NetMF by orders of magnitude for large-scale networks, it also has competitive, and sometimes better, performance. This demonstrates the effectiveness of our spectral sparsification based approximation algorithm.

NetSMF vs. DeepWalk, LINE & node2vec. We also compare NetSMF against common graph embedding benchmarks—DeepWalk, LINE, and node2vec. For the OAG dataset, DeepWalk and node2vec fail to finish the computation within one week, while NetSMF requires only 24 hours. Based on the publicly reported running time of skip-gram [24], we estimate that DeepWalk and node2vec may require months to generate embeddings for the OAG dataset. In BlogCatalog, DeepWalk and NetSMF require similar computing time, while in Flickr, YouTube, and PPI, NetSMF is 2.75×, 5.9×, and 24× faster than DeepWalk, respectively. In all the datasets, NetSMF achieves 4–24× speedup over node2vec.

Moreover, the performance of NetSMF is significantly better than DeepWalk in BlogCatalog, PPI, and Flickr, by 7–34% in terms of Micro-F1 and 5–25% in terms of Macro-F1. In YouTube, NetSMF achieves comparable results to DeepWalk. Compared with node2vec, NetSMF achieves comparable performance in BlogCatalog and YouTube, and significantly better performance in PPI and Flickr. In summary, NetSMF consistently outperforms DeepWalk and node2vec in terms of both efficiency and effectiveness.

LINE has the best efficiency among all the five methods and together with NetSMF, they are the only methods that can generate embeddings for OAG within one week (and both finish in one day). However, it also has the worst prediction performance and consistently loses to others by a large margin across all datasets. For example, NetSMF beats LINE by 21% and 39% in Flickr, and by 30% and 100% in OAG in terms of Micro-F1 and Macro-F1, respectively.

In summary, LINE achieves efficiency at the cost of ignoring multi-hop dependencies in networks, which are supported by all the other four methods—DeepWalk, node2vec, NetMF, and NetSMF, demonstrating the importance of multi-hop dependencies for learning network representations.

More importantly, among these four methods, DeepWalk achieves neither efficiency nor effectiveness superiority; node2vec achieves relatively good performance at the cost of efficiency; NetMF achieves effectiveness at the expense of significantly increased time and space costs; NetSMF is the only method that achieves both high efficiency and effectiveness, empowering it to learn effective embeddings for billion-scale networks (e.g., the OAG network with 0.9 billion edges) in one day on one modern server.

4.4 Parameter Analysis

In this section, we discuss how the hyper-parameters influence the performance and efficiency of NetSMF. We report all the parameter analyses on the Flickr dataset with training ratio set to be 10%.

How to Set the Embedding Dimension d. As mentioned in Section 3.1, SVD allows us to determine a "good" embedding dimension without supervised information. There are many methods available such as captured energy and Cattell's Scree test [5]. Here we propose to use Cattell's Scree test. Cattell's Scree test plots the singular values and selects a rank d such that there is a clear drop in the magnitudes or the singular values start to even out. In Flickr, if we sort the singular values in decreasing order, we can observe that the singular values approach 0 when the rank increases to around 100, as shown in Figure 3(b). In our experiments, by varying d form 2^4 to 2^8, we reach the best performance at $d = 128$, as shown in Figure 3(a), demonstrating the ability of our matrix factorization based NetSMF for automatically determining the embedding dimension.

The Number of Non-Zeros M. In theory, $M = O(Tm\epsilon^{-2}\log n)$ is required to guarantee the approximation error (See Section 3.1). Without loss of generality, we empirically set M to be $k \times T \times m$ where k is chosen from 1, 10, 100, 200, 500, 1000, 2000 and investigate how the number of non-zeros influence the quality of learned embeddings. As shown in Figure 3(c), when increasing the number of non-zeros, NetSMF tends to have better prediction performance because the original matrix is being approximated more accurately. On the other hand, although increasing M has a positive effect on the prediction performance, its marginal benefit diminishes gradually. One can observe that setting $M = 1000 \times T \times m$ (the second-to-the-right data point on each line in Figure 3(c)) is a good choice that balances NetSMF's efficiency and effectiveness.

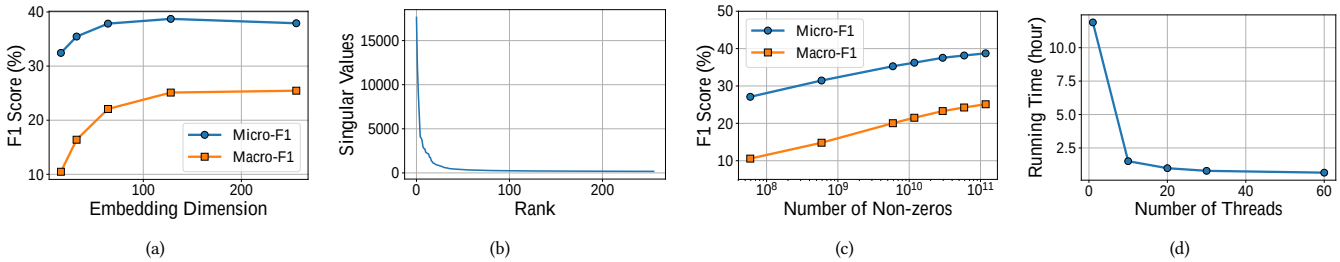
The Number of Threads. In this work, we use a single-machine shared memory implementation with multi-threading acceleration. We report the running time of NetSMF when setting the number of threads to be 1, 10, 20, 30, 60, respectively. As shown in Figure 3(d), NetSMF takes 12 hours to embed the Flickr network with one thread and 48 minutes to run with 30 threads, achieving a 15× speedup ratio (with ideal being 30×). This relatively good sub-linear speedup supports NetSMF to scale up to very large-scale networks.

5 RELATED WORK

In this section, we review the related work of network embedding, large-scale embedding algorithms, and spectral graph sparsification.

5.1 Network Embedding

Network embedding has been extensively studied over the past years [16]. The success of network embedding has driven a lot of downstream network applications, such as recommendation systems [44]. Briefly, recent work about network embedding can be categorized into three genres: (1) Skip-gram based methods that are inspired by word2vec [24], such as LINE [33], DeepWalk [27], node2vec [14], metapath2vec [12], and VERSE [40]; (2) Deep learning based methods such as [21, 44]; (3) Matrix factorization based methods such as GraRep [4] and NetMF [28]. Among them, NetMF bridges the first and the third categories by unifying a collection of skip-gram based network embedding methods into a matrix factorization framework. In this work, we leverage the merit of NetMF and address its limitation in efficiency. Among literature, PinSage is notably a network embedding framework for billion-scale networks [44]. The difference between NetSMF and PinSage



(a) (b) (c) (d)

Figure 3: Parameter analysis: (a) Prediction performance v.s. embedding dimension d; (b) Cattel's Scree Test on singular values. (c) Prediction performance v.s. the number of non-zeros M; (d) Running time v.s. the number of threads.

lies in the following aspect. The goal of NetSMF is to pre-train general network embeddings in an unsupervised manner, while PinSage is a supervised graph convolutional method with both the objective of recommender systems and existing node features incorporated. That being said, the embeddings learned by NetSMF can be consumed by PinSage for downstream network applications.

5.2 Large-Scale Embedding Learning

Studies have attempted to optimize embedding algorithms for large datasets from different perspectives. Some focus on improving skip-gram model, while others consider it as matrix factorization.

Distributed Skip-Gram Model. Inspired by word2vec [25], most of the modern embedding learning algorithms are based on the skip-gram model. There is a sequence of work trying to accelerate the skip-gram model in a distributed system. For example, Ji et al. [19] replicate the embedding matrix on multiple workers and synchronize them periodically; Ordentlich et al. [26] distribute the columns (dimensions) of the embedding matrix to multiple executors and synchronize them with a parameter server [23]. Negative sampling is a key step in skip-gram, which requires to draw samples from a noisy distribution. Stergiou et al. [32] focus on the optimization of negative sampling by replacing the roulette wheel selection with a hierarchical sampling algorithm based on the alias method. More recently, Wang et al. [43] propose a billion-scale network embedding framework by heuristically partitioning the input graph to small subgraphs, and processing them separately in parallel. However, the performance of their framework highly relies on the quality of graph partition. The drawback for partition-based embedding learning is that the embeddings learned in different subgraphs do not share the same latent space, making it impossible to compare nodes across subgraphs.

Efficient Matrix Factorization. Factorizing the NetMF matrix, either implicitly (e.g., LINE [33] and DeepWalk [27]) or explicitly (e.g., NetMF [28]), encounters two issues. First, the denseness of this matrix makes computation expensive even for a moderate context window size (e.g., $T = 10$). Second, the non-linear transformation, i.e., element-wise matrix logarithm, is hard to approximate. LINE [33] solves this problem by setting $T = 1$. With such simplification, it achieves good scalability at the cost of prediction performance. NetSMF addresses these issues by efficiently sparsifying the dense NetMF matrix with a theoretically-bounded approximation error.

5.3 Spectral Graph Sparsification

Spectral graph sparsification has been studied for decades in graph theory [38]. The task of graph sparsification is to approximate a "dense" graph by a "sparse" one that can be effectively used in place of the dense one [38], which arises in many applications such as scientific computing [17], machine learning [3, 7] and data mining [46]. Our NetSMF model is the first work that incorporates spectral sparsification algorithms [7, 8] into network embedding, which offers a powerful and efficient way to approximate and analyze the random-walk matrix-polynomial in the NetMF matrix.

6 CONCLUSION

In this work, we study network embedding with the goal of achieving both efficiency and effectiveness. To address the scalability challenges faced by the NetMF model, we propose to study large-scale network embedding as sparse matrix factorization. We present the NetSMF algorithm, which achieves a sparsification of the (dense) NetMF matrix. Both the construction and factorization of the sparsified matrix are fast enough to support very large-scale network embedding learning. For example, it empowers NetSMF to efficiently embed the Open Academic Graph in 24 hours, whose size is computationally intractable for the dense matrix factorization solution (NetMF). Theoretically, the sparsified matrix is spectrally close to the original NetMF matrix with an approximation bound. Empirically, our extensive experimental results show that the sparsely learned embeddings by NetSMF are as effective as those from the factorization of the NetMF matrix, leaving it outperform the common network embedding benchmarks—DeepWalk, LINE, and node2vec. In other words, among both matrix factorization based methods (NetMF and NetSMF) and common skip-gram based benchmarks (DeepWalk, LINE, and node2vec), NetSMF is the only model that achieves both efficiency and performance superiority.

Future Work. NetSMF brings an efficient, effective, and guaranteed solution to network embedding learning. There are multiple tangible research fronts we can pursue. First, our current single-machine implementation limits the number of samples we can take for large networks. We plan to develop a multi-machine solution in the future to further scale NetSMF. Second, building upon NetSMF, we would like to efficiently and accurately learn embeddings for large-scale directed [9], dynamic [20], and/or heterogeneous networks. Third, as the advantage of matrix factorization methods demonstrated, we are also interested in exploring the other matrix

definitions that may be effective in capturing different structural properties in networks. Last, it would be also interesting to bridge matrix factorization based network embedding methods with graph convolutional networks.

Acknowledgements. We would like to thank Dehua Cheng and Youwei Zhuo from USC for helpful discussions. Jian Li is supported in part by the National Basic Research Program of China Grant 2015CB358700, the National Natural Science Foundation of China Grant 61822203, 61772297, 61632016, 61761146003, and a grant from Microsoft Research Asia. Jie Tang is the corresponding author.

APPENDIX

We first prove Thm. 2 and Thm. 3 in Section 3.3. The following lemmas will be useful in our proof.

Lemma 1. *([39]) Singular values of a real symmetric matrix are the absolute values of its eigenvalues.*

Lemma 2. *(Courant-Fisher Theorem) Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, then for $i \in [n]$,*

$$\lambda_i = \min_{\dim(U)=i} \max_{x\in U; \|x\|_2=1} x^\top A x.$$

Lemma 3. *([18]) Let B, C be two $n \times n$ symmetric matrices. Then for the decreasingly-ordered singular values σ of B, C and BC,*

$$\sigma_{i+j-1}(BC) \leq \sigma_i(B) \times \sigma_j(C)$$

holds for any $1 \leq i, j \leq n$ and $i + j \leq n + 1$.

Lemma 4. *Let $\mathcal{L} = D^{-1/2}LD^{-1/2}$ and similarly $\widehat{\mathcal{L}} = D^{-1/2}\widehat{L}D^{-1/2}$. Then all the singular values of $\widehat{\mathcal{L}} - \mathcal{L}$ are smaller than 2ϵ, i.e., $\forall i \in [n]$, $\sigma_i(\widehat{\mathcal{L}} - \mathcal{L}) < 4\epsilon$.*

Proof. Notice that

$$\mathcal{L} = D^{-1/2}LD^{-1/2} = I - \sum_{r=1}^{T} \alpha_r \left(D^{-1/2}AD^{-1/2}\right)^r$$

which is a normalized graph Laplacian whose eigenvalues lie in the interval $[0, 2)$, i.e., for $i \in [n]$, $\lambda_i(\mathcal{L}) \in [0, 2)$ [42]. Since $\widehat{L}$ is a $(1 + \epsilon)$-spectral sparsifier of L, we know that for $\forall x \in \mathbb{R}^n$,

$$\frac{1}{1+\epsilon} x^\top L x \leq x^\top \widehat{L} x \leq \frac{1}{1-\epsilon} x^\top L x.$$

Let $x = D^{-1/2}y$ which is bijective, we have

$$\frac{1}{1+\epsilon} y^\top \mathcal{L} y \leq y^\top \widehat{\mathcal{L}} y \leq \frac{1}{1-\epsilon} y^\top \mathcal{L} y$$
$$\Longrightarrow y^\top(\widehat{\mathcal{L}} - \mathcal{L})y \leq \frac{\epsilon}{1-\epsilon} y^\top \mathcal{L} y < 2\epsilon y^\top \mathcal{L} y.$$

The last inequality is because we assume $\epsilon < 0.5$. Then, by Courant-Fisher Theorem (Lemma 2), we can immediately get, $\forall i \in [n]$,

$$\lambda_i(\widehat{\mathcal{L}} - \mathcal{L}) \leq 2\epsilon\lambda_i(\mathcal{L}) < 4\epsilon.$$

Then, by Lemma 1, $\sigma_i(\widehat{\mathcal{L}} - \mathcal{L}) < 4\epsilon, \forall i \in [n]$.

Given the above lemmas, we can see how the constructed $\widehat{M}$ approximates M and how the constructed NetMF matrix sparsifier (Eq. (6)) approximates the NetMF matrix (Eq. (3)).

Theorem 2. *The singular value of $\widehat{M} - M$ satisfies $\sigma_i(\widehat{M} - M) \leq \frac{4\epsilon}{\sqrt{d_i d_{\min}}}, \forall i \in [n]$.*

Proof. First notice that $\widehat{M} - M = D^{-1}\left(\widehat{L} - L\right)D^{-1} = D^{-1/2}(\widehat{\mathcal{L}} - \mathcal{L})D^{-1/2}$. Apply Lemma 3 twice and use the result from Lemma 4, we have

$$\sigma_i\left(\widehat{M} - M\right) \leq \sigma_i\left(D^{-1/2}\right) \times \sigma_1\left(\widehat{\mathcal{L}} - \mathcal{L}\right) \times \sigma_1\left(D^{-1/2}\right)$$
$$\leq \frac{1}{\sqrt{d_i}} \times 4\epsilon \times \frac{1}{\sqrt{d_{\min}}} = \frac{4\epsilon}{\sqrt{d_i d_{\min}}}.$$

Theorem 3. *Let $\|\cdot\|_F$ be the matrix Frobenius norm. Then*

$$\left\|\text{trunc\_log}^\circ\left(\frac{\text{vol}(G)}{b}\widehat{M}\right) - \text{trunc\_log}^\circ\left(\frac{\text{vol}(G)}{b}M\right)\right\|_F \leq \frac{4\epsilon\,\text{vol}(G)}{b\sqrt{d_{\min}}}\sqrt{\sum_{i=1}^{n}\frac{1}{d_i}}.$$

Proof. It is easy to observe that trunc_log$^\circ$ is 1-Lipchitz w.r.t. Frobenius norm. So we have

$$\left\|\text{trunc\_log}^\circ\left(\frac{\text{vol}(G)}{b}\widehat{M}\right) - \text{trunc\_log}^\circ\left(\frac{\text{vol}(G)}{b}M\right)\right\|_F$$
$$\leq \left\|\frac{\text{vol}(G)}{b}\widehat{M} - \frac{\text{vol}(G)}{b}M\right\|_F = \frac{\text{vol}(G)}{b}\left\|\widehat{M} - M\right\|_F$$
$$= \frac{\text{vol}(G)}{b}\sqrt{\sum_{i\in[n]}\sigma_i^2(\widehat{M} - M)} \leq \frac{4\epsilon\,\text{vol}(G)}{b\sqrt{d_{\min}}}\sqrt{\sum_{i=1}^{n}\frac{1}{d_i}}.$$

We finally explain the remaining question in Step 1 of NetSMF: After sampling a length-r path $p = (u_0, \cdots, u_r)$, why does the algorithm add a new edge to the sparsifier with weight $\frac{rm}{MZ(p)}$? Our proof relies on two lemmas from [8].

Lemma 5. *(Lemma 3.3 in [8]) Given the path length r, the probability for the PathSampling algorithm to sample a path p is $\tau(p) = \frac{w(p)Z(p)}{2rm}$, where $Z(p)$ is defined in Eq. (7) and*

$$w(p) = \frac{\prod_{i=1}^{r} A_{u_{i-1},u_i}}{\prod_{i=1}^{r-1} D_{u_i}}.$$

Lemma 6. *(Theorem 2.2 in [8]) After sampling a length-r path $p = (u_0, u_1, \cdots, u_r)$, the weight corresponding to the new edge (u_0, u_r) added to the sparsifier should be $\frac{w(p)}{\tau(p)M}$.*

Theorem 4. *After sampling a length-r path $p = (u_0, u_1, \cdots, u_r)$ using the PathSampling algorithm (Alg. 2). The weight of the new edge added to the sparsifier $\widehat{L}$ is $\frac{2rm}{MZ(p)}$.*

Proof. The proof is to plug the definition of $Z(p)$, $w(p)$, and $\tau(p)$ from Lemma 5 into Lemma 6, that is,

$$\frac{w(p)}{\tau(p)M} = \frac{w(p)}{\frac{w(p)Z(p)}{2rm} \times M} = \frac{2rm}{MZ(p)}.$$

For unweighted networks, this weight can be simplified to $\frac{m}{M}$, since $Z(p) = 2r$ for unweighted networks.

REFERENCES

[1] Nitin Agarwal, Huan Liu, Sudheendra Murthy, Arunabha Sen, and Xufei Wang. 2009. A Social Identity Approach to Identify Familiar Strangers in a Social Network.. In *ICWSM '09*.

[2] Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. 2012. Four degrees of separation. In *WebSci '12*. ACM, 33–42.

[3] Daniele Calandriello, Ioannis Koutis, Alessandro Lazaric, and Michal Valko. 2018. Improved large-scale graph learning through ridge spectral sparsification. In *ICML '18*. 687âĂŞ696.

[4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning graph representations with global structural information. In *CIKM '15*. ACM, 891–900.

[5] Raymond B Cattell. 1966. The scree test for the number of factors. *Multivariate behavioral research* 1, 2 (1966), 245–276.

[6] Kamalika Chaudhuri, Fan Chung, and Alexander Tsiatas. 2012. Spectral clustering of graphs with general degrees in the extended planted partition model. In *COLT '12*. 35–1.

[7] Dehua Cheng, Yu Cheng, Yan Liu, Richard Peng, and Shang-Hua Teng. 2015. Efficient sampling for Gaussian graphical models via spectral sparsification. In *COLT '15*. 364–390.

[8] Dehua Cheng, Yu Cheng, Yan Liu, Richard Peng, and Shang-Hua Teng. 2015. Spectral sparsification of random-walk matrix polynomials. *arXiv preprint arXiv:1502.03496* (2015).

[9] Michael B Cohen, Jonathan Kelner, John Peebles, Richard Peng, Aaron Sidford, and Adrian Vladu. 2016. Faster algorithms for computing the stationary distribution, simulating random walks, and more. In *FOCS '16*. IEEE, 583–592.

[10] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering* 5, 1 (1998), 46–55.

[11] Anirban Dasgupta, John E Hopcroft, and Frank McSherry. 2004. Spectral analysis of random graphs with skewed degree distributions. In *FOCS '04*. 602–610.

[12] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *KDD '17*.

[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. *JMLR '08* 9, Aug (2008), 1871–1874.

[14] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD '16*. ACM, 855–864.

[15] 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.

[16] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data(base) Engineering Bulletin* 40 (2017), 52–74.

[17] Nicholas J Higham and Lijing Lin. 2011. On p th roots of stochastic matrices. *Linear Algebra Appl.* 435, 3 (2011), 448–463.

[18] Roger A. Horn and Charles R. Johnson. 1991. *Topics in Matrix Analysis*. Cambridge University Press. https://doi.org/10.1017/CBO9780511840371

[19] Shihao Ji, Nadathur Satish, Sheng Li, and Pradeep Dubey. 2016. Parallelizing word2vec in shared and distributed memory. *arXiv preprint arXiv:1604.04661* (2016).

[20] Michael Kapralov, Yin Tat Lee, CN Musco, CP Musco, and Aaron Sidford. 2017. Single pass spectral sparsification in dynamic streams. *SIAM J. Comput.* 46, 1 (2017), 456–477.

[21] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR '17*.

[22] Omer Levy and Yoav Goldberg. 2014. Neural Word Embedding as Implicit Matrix Factorization. In *NIPS '14*. 2177–2185.

[23] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server.. In *OSDI '14*, Vol. 14. 583–598.

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *ICLR Workshop '13*.

[25] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS' 13*. 3111–3119.

[26] Erik Ordentlich, Lee Yang, Andy Feng, Peter Cnudde, Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, and Gavin Owens. 2016. Network-efficient distributed word2vec training system for large vocabularies. In *CIKM '16*. ACM, 1139–1148.

[27] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD '14*. ACM, 701–710.

[28] 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 *WSDM '18*. ACM, 459–467.

[29] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-june Paul Hsu, and Kuansan Wang. 2015. An overview of microsoft academic service (mas) and applications. In *WWW '15*. ACM, 243–246.

[30] Daniel A Spielman and Nikhil Srivastava. 2011. Graph sparsification by effective resistances. *SIAM J. Comput.* 40, 6 (2011), 1913–1926.

[31] Chris Stark, Bobby-Joe Breitkreutz, Andrew Chatr-Aryamontri, Lorrie Boucher, Rose Oughtred, Michael S Livstone, Julie Nixon, Kimberly Van Auken, Xiaodong Wang, Xiaoqi Shi, et al. 2010. The BioGRID interaction database: 2011 update. *Nucleic acids research* 39, suppl_1 (2010), D698–D704.

[32] Stergios Stergiou, Zygimantas Straznickas, Rolina Wu, and Kostas Tsioutsiouliklis. 2017. Distributed Negative Sampling for Word Embeddings.. In *AAAI '17*. 2569–2575.

[33] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW '15*. 1067–1077.

[34] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *KDD '08*. 990–998.

[35] Lei Tang and Huan Liu. 2009. Relational learning via latent social dimensions. In *KDD '09*. ACM, 817–826.

[36] Lei Tang and Huan Liu. 2009. Scalable learning of collective behavior based on sparse social dimensions. In *CIKM '09*. ACM, 1107–1116.

[37] Lei Tang, Suju Rajan, and Vijay K Narayanan. 2009. Large scale multi-label classification via metalabeler. In *WWW '09*. ACM, 211–220.

[38] Shang-Hua Teng et al. 2016. Scalable algorithms for data and network analysis. *Foundations and Trends® in Theoretical Computer Science* 12, 1–2 (2016), 1–274.

[39] Lloyd N Trefethen and David Bau III. 1997. *Numerical linear algebra*. Vol. 50. Siam.

[40] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. VERSE: Versatile Graph Embeddings from Similarity Measures. In *WWW '18*. 539–548.

[41] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. 2009. Mining multi-label data. In *Data mining and knowledge discovery handbook*. Springer, 667–685.

[42] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and computing* 17, 4 (2007), 395–416.

[43] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba. In *KDD '18*. ACM.

[44] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *KDD '18*.

[45] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. *HotCloud '10* 10, 10-10 (2010), 95.

[46] Peixiang Zhao. 2015. gSparsify: Graph Motif Based Sparsification for Graph Clustering. In *CIKM '15*. ACM, 373–382.