Learning Community Embedding with Community Detection and Node Embedding on Graphs

Sandro Cavallari
Nanyang Technological University
Singapore
sandro001@e.ntu.edu.sg

Vincent W. Zheng
Advanced Digital Sciences Center
Singapore
vincent.zheng@adsc.com.sg

Hongyun Cai
Advanced Digital Sciences Center
Singapore
hongyun.c@adsc.com.sg

Kevin Chen-Chuan Chang
University of Illinois at Urbana-Champaign
IL, USA
kcchang@illinois.edu

Erik Cambria
Nanyang Technological University
Singapore
cambria@ntu.edu.sg

ABSTRACT

In this paper, we study an important yet largely under-explored setting of graph embedding, i.e., embedding communities instead of each individual nodes. We find that community embedding is not only useful for community-level applications such as graph visualization, but also beneficial to both community detection and node classification. To learn such embedding, our insight hinges upon a closed loop among community embedding, community detection and node embedding. On the one hand, node embedding can help improve community detection, which outputs good communities for fitting better community embedding. On the other hand, community embedding can be used to optimize the node embedding by introducing a community-aware high-order proximity. Guided by this insight, we propose a novel community embedding framework that jointly solves the three tasks together. We evaluate such a framework on multiple real-world datasets, and show that it improves graph visualization and outperforms state-of-the-art baselines in various application tasks, e.g., community detection and node classification.

CCS CONCEPTS

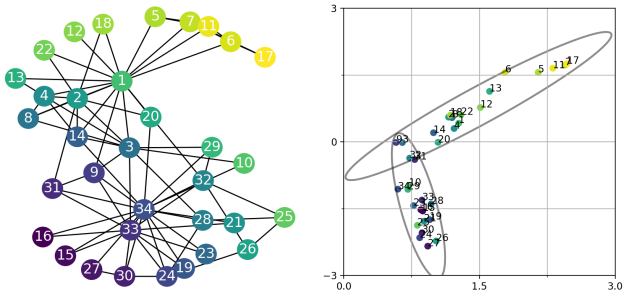
• **Computing methodologies** → **Neural networks**; *Machine learning algorithms*; • **Mathematics of computing** → *Probabilistic algorithms*; • **Applied computing** → *Sociology*;

KEYWORDS

community embedding, graph embedding

1 INTRODUCTION

Traditionally, graph embedding focuses on individual *nodes* and aims to output a vector representation for each node in the graph, such that two nodes "close" on the graph have similar vector representations in a low-dimensional space. Such node embedding has been shown very successful in preserving the network structure, and significantly improving a wide range of applications, including node classification [5, 20], node clustering [27, 34], link prediction [12, 19], graph visualization [24, 29] and more [10, 18].



(a) Karate club graph (b) Visualization based on our model

Figure 1: Embedding nodes and communities in a 2D space.

In this paper, we study another important, yet largely under-explored setting of graph embedding, which focuses on embedding *communities*. Generally, a "community embedding" is a representation for a community in a low-dimensional space. Because a community is a group of densely connected nodes, a community embedding is expected to characterize how its member nodes distribute in the low-dimensional space. As a result, we cannot simply define a community embedding as a vector; instead, we need to define it as a *distribution* in the low-dimensional space. In Fig. 1, we use the well-studied Karate Club graph[1] as an example to demonstrate community embedding in a 2D space. As shown in Fig. 1(a), the Karate Club graph has 34 nodes and 78 edges. It is known that this graph has two communities, one of which is led by a class instructor (node 1) and the other of which is led by a club administrator (node 34) [35]. Some club members (e.g., node 9) are identified as "weak supporters" to the two communities, thus they can belong to both. In Fig. 1(b), we visualize the graph in the 2D space, where each node embedding is a 2D vector.

[1] https://networkdata.ics.uci.edu/data.php?id=105

CIKM'17 , November 6–10, 2017, Singapore, Singapore

ACM ISBN 978-1-4503-4918-5/17/11…$15.00
https://doi.org/10.1145/3132847.3132925

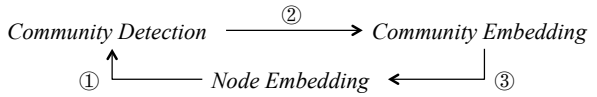


Figure 2: A closed loop for learning community embedding.

Because each community is a group of densely connected nodes, we are motivated by the Gaussian mixture model (GMM) [3] to see each community embedding as a *multivariate Gaussian distribution* in the 2D space. Consequently, we visualize the two overlapping communities in the Karate Club graph as two overlapping eclipses, each of which is characterized by a 2D mean vector and a 2×2 covariance matrix. Community embedding is useful for many community-level applications, e.g., for community visualization to help generate insights from big graphs, or community recommendation to search for similar communities.

Learning community embedding is non-trivial. On the one hand, to have meaningful community embedding, we first need to well identify the communities. Then, a straightforward approach for community embedding is to: (1) run community detection, such as Spectral Clustering [25], on the graph to get community assignments for each node; (2) apply node embedding, such as DeepWalk [20] or LINE [24], on the graph to get an embedding vector for each node; (3) aggregate the node embedding vectors in each community, so as to fit a (multivariate Gaussian) distribution as its community embedding. Such a pipeline approach is suboptimal, because its community detection is independent of its node embedding. On the other hand, recent studies show that node embedding often improves community detection, thanks to its well preserving the network structure in a low-dimensional space [5, 15, 27]. Hence, another possible approach for community embedding is to directly run community detection over the node embedding results and, hence, fit a (multivariate Gaussian) distribution for each community based on its node embedding vectors. However, such an approach is also suboptimal, because most of the existing node embedding methods (e.g., DeepWalk [20], LINE[24] and Node2Vec [12]) are not aware of community structure, which makes their node embedding inputs suboptimal for the subsequent community detection. In Sec. 5, we empirically evaluate both of the above approaches, and show that their performances are limited. There is few work that considers node embedding and community detection together; they either require extra supervision (e.g., must-links) [34] or high computational complexity (e.g., quadratic to the number of nodes in a graph) [31]. Moreover, they under-characterize a community in the low-dimensional space as a vector, thus it is difficult to accurately visualize overlapping communities.

Our insight for learning community embedding is that, there exists a closed loop among community detection, community embedding and node embedding, as shown in Fig. 2. On the one hand, as discussed earlier, node embedding can help improve community detection (i.e., ①), which outputs good communities for fitting meaningful community embedding (i.e., ②). On the other hand, community embedding can be used to optimize node embedding (i.e., ③). Suppose for a community k, we already have its community embedding as a multivariate distribution in a low-dimensional space. Then, we can enforce community k's member nodes to scatter closely near its community embedding's mean vector in that low-dimensional space. As a result, these same-community nodes tend to have similar node embedding vectors. Compared with first- and second-order proximity, community embedding does not require two nodes to be directly linked or share many "contexts" for being close. Because the connections between two nodes in a community can be high-order, we consider community embedding as introducing a *community-aware high-order proximity* to node embedding. This feedback from community embedding to node embedding helps us to close the loop; hopefully the community-aware node embedding can serve as better inputs for the subsequent community detection, thus leading to more meaningful community embedding results.

Guided by the closed loop insight, we propose *ComE*, a novel Community Embedding framework that jointly solves community embedding, community detection and node embedding together. We define community embedding as a multivariate Gaussian distribution, and use it to empower community detection from the node embedding results by a Gaussian mixture formulation. Denote a graph as $G = (V, E)$, where V is the set of nodes and E is the set of edges. This Gaussian mixture formulation enables us to efficiently detect the communities and infer their community embedding distributions from G in $O(|V|)$ time. Given community assignments and community embedding, we extend the neural network formulations of DeepWalk and LINE to preserve first-, second- and high-order (community-aware) proximity together. For this neural network formulation, we propose a scalable inference algorithm, whose complexity is linear to the graph size $O(|V| + |E|)$.

We summarize our contributions as follows.

- We introduce a novel joint modeling framework, which leverages the closed loop among node embedding, community detection and community embedding, to learn graph embedding.
- We contribute with a scalable inference algorithm which complexity of $O(|V| + |E|)$, is often lower than the existing higher-order proximity-aware methods (Tab. 1).
- We evaluate ComE on multiple real-world datasets with various application tasks. It renders better graph visualization results, and also improves the state-of-the-art baselines by at least 6.6% (NMI) and 2.2%–16.9% (conductance) in community detection, 0.8%–26.9% (macro-F1) and 0.71%–48% (micro-F1) in node classification.

2 RELATED WORK

We summarize the differences of our work with some representative related work on graph embedding and community detection in Tab. 1. Next we will detail the discussion of related work.

2.1 Graph Embedding

As there is an increasing amount of graph data, ranging from social networks to various information networks, an important question arises is how to represent a graph for analytics [11]. Graph embedding is the state-of-the-art graph representation framework, which aims to project a graph into a low-dimensional space for further applications [16, 20, 26]. In terms of the target to embed, most of the existing graph embedding methods focus on nodes. For example, earlier methods, such as MDS [7], LLE [21], IsoMap [26] and Laplacian eigenmap [2], aim to preserve the first-order proximity extracting the leading eigenvectors of a graph affinity matrices.

Table 1: Comparison with related work. In the following analysis are reported only factors dependent to the graph.

	node embed.	community embed.	community detection	model complexity
DeepWalk [20]	•			$O(\|V\| \log \|V\|)$
LINE [24]	•			$O(a\|E\|)$
Node2Vec [12]	•			$O(\|V\| \log \|V\| + \|V\|a^2)$
GraRep [5]	•			$O(\|V\|^3)$
Spectral [25]			•	$O(\|V\|^3)$
DNR [34]	•		•	$O(\|V\|^2)$
M-NMF [31]	•		•	$O(\|V\|^2)$
ComE	•	•	•	$O(\|V\| + \|E\|)$

More recent methods start to exploit neural networks to learn the representation for each node, with either shallow architectures [12, 24, 33] or deep architectures [1, 8, 18, 29]. DeepWalk [20] models the second-order proximity for node embedding with path sampling, and its complexity using hierarchical softmax[2] for inference is $O(|V| \log |V|)$. Node2Vec [12] extends DeepWalk with a controlled path sampling process, which requires $O(|V|a^2)$ where a is the average degree of the graph; thus, its model complexity is $O(|V| \log |V| + |V|a^2)$. LINE [24] and SDNE [29] preserve both first- and second-order proximity at the price of a higher complexity, respectively $O(a|E|)$ and $O(a|V|)$.

Compared with our methods, the above works have a lower or comparable complexity, but neither try to detect nor represent the communities. Community structure is known as an important network property, and it has been considered in node embedding. For example, in SAE [27], the authors show that spectral clustering can be regarded as reconstructing a graph's normalized similarity matrix, but it is expensive with a complexity of at least $O(|V|^{2.367})$. Thus, they propose to directly construct the normalized similarity matrix with $O(|E|)$ complexity, and input it to stacked Auto-Encoder for reconstruction with $O(|V|)$ complexity. The resulting node embedding is used for K-means clustering and is shown to obtain better communities than spectral clustering. Similarly, DNR [34] constructs a modularity matrix from the graph with $O(|V|^2)$ complexity, then applies stacked Auto-Encoder to the modularity matrix for node embedding. It also introduces must-links to supervise the node embedding.

Higher-order of proximity methods, such as GraRep [5] and HOPE [19], are not explicitly community aware. Besides this, GraRep learn a high-order transition probability matrices and later run Singular Value Decomposition (SVD) for a $O(|V|^3)$ complexity. The above-mentioned neither tries to embed communities, nor explicitly detects communities in node embedding, but generally have a higher complexity due to the sparsity of real work network. There is little work that tries to explicitly embed communities in a low-dimensional space. For example, M-NMF [31] constructs the modularity matrix with $O(|V|^2)$ complexity, then applies non-negative matrix factorization to learn node embedding and community detection together with a complexity proportional to $O(|V|^2)$.

Comparatively, M-NMF represents each community with a vector, thous we would not consider it to produce a community embedding; besides its complexity is generally higher than our complexity of $O(|V| + |E|)$, since in practice the graphs are sparse with $|E| \ll |V|^2$.

[2]If using negative sampling, the complexity becomes $O(|V|)$.

Table 2: Notations used in this paper.

Notation	Description
$G(V, E)$	Graph G, nodes V and edges E
ℓ	Length of each random walk path in sampling
γ	Number of random walks for each node in sampling
ζ	Context size
m	Negative context size
$\boldsymbol{\phi}_i \in \mathbb{R}^d$	Node embedding of node i
$\boldsymbol{\phi}'_i \in \mathbb{R}^d$	Context embedding of node i
$\mathcal{N}(\boldsymbol{\psi}_k, \Sigma_k)$	Community embedding of community k
$\boldsymbol{\psi}_k \in \mathbb{R}^d$	Community embedding k's mean vector
$\Sigma_k \in \mathbb{R}^{d \times d}$	Community embedding k's covariance matrix
$\pi_{ik} \in [0, 1]$	Community membership of node i to community k
$P_n(\cdot)$	Negative sampling probability
K	Number of communities on G
α	Trade-off parameter for context embedding
β	Trade-off parameter for community embedding
a	graph's average degree

2.2 Community Detection

Community detection aims to discover groups of nodes on a graph, such that the intra-group connections are denser than the inter-group ones [30]. With the prevalence of social networks, recent community detection studies start to exploit rich node interactions on the graphs, such as nodes with content [22], attributes [23] and node-to-node diffusion [4]. A comprehensive survey of recent community detection algorithms can be found in [32]. In this work, our community detection is applied to homogeneous graphs, whose nodes and edges do not have additional information. Earlier community detection methods on homogeneous graphs often apply different clustering algorithms directly on the graph adjacency matrix. For example, in [25], spectral clustering is applied to the social networks for extracting the communities. In [13], a Laplacian Regularized GMM is trained to capture the manifold structure of a nearest neighbor graph.

With the recent development of neural networks and deep learning, node embedding is utilized to assist community detection [15, 27]. Such work usually first embeds the graph in a low-dimensional space, and then apply clustering algorithms such as K-means on the embedding results. Despite the success of these node embedding based methods in detecting communities, they often do not jointly optimize node embedding and community detection together. As their goals are mainly community detection, they do not necessarily have an explicit notion of community embedding.

3 PROBLEM FORMULATION

As *input*, we are given a graph $G = (V, E)$, where V is the node set and E is the edge set. Traditional graph embedding aims to learn a node embedding for each $v_i \in V$ as $\boldsymbol{\phi}_i \in \mathbb{R}^d$. In this paper, we also try to learn community embedding. Suppose there are K communities on the graph G. For each node v_i, we denote its community assignment as $z_i \in \{1, ..., K\}$.

Motivated by the Gaussian mixture formulation [3], we define community embedding as a multivariate Gaussian distribution in a low-dimensional space.

Definition 3.1. **Community embedding** of a community k (with $k \in \{1, ..., K\}$) in a d-dimensional space is a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\psi}_k, \Sigma_k)$, where $\boldsymbol{\psi}_k \in \mathbb{R}^d$ is a mean vector and $\Sigma_k \in \mathbb{R}^{d \times d}$ is a covariance matrix.

As *output*, we aim to learn: (1) node embedding $\boldsymbol{\phi}_i$ for each node $v_i \in V$; (2) community membership π_{ik}, such that $\sum_{k=1}^{K} \pi_{ik} = 1$, for each node $v_i \in V$ and each community $k \in \{1, ..., K\}$; (3) community embedding parameters $(\boldsymbol{\psi}_k, \Sigma_k)$ for each community $k \in \{1, ..., K\}$.

We summarize all of our notations in Tab. 2.

3.1 Community Detection and Embedding

Given node embedding, one straightforward way to detect communities and learn their community embedding is to take a pipeline approach. For example, as shown in Fig. 2, we can run Spectral Clustering to detect communities, then fit a Gaussian mixture for each community. However, such a pipeline approach lacks a unified objective function, thus, being hard to optimize later with node embedding. Alternatively, we can do community detection and embedding together in one single objective function based on GMM. That is, we consider each node v_i's embedding $\boldsymbol{\phi}_i$ as generated by a multivariate Gaussian distribution from a community $z_i = k$. Then, for all the nodes in V, we have the likelihood as

$$\prod_{i=1}^{|V|} \sum_{k=1}^{K} p(z_i = k) p(v_i | z_i = k; \boldsymbol{\phi}_i, \boldsymbol{\psi}_k, \Sigma_k) \tag{1}$$

where $p(z_i = k)$ is the probability of node v_i belonging to community k. For notation simplicity, we denote $p(z_i = k)$ as π_{ik}; thus, we have $\pi_{ik} \in [0, 1]$ and $\sum_{k=1}^{K} \pi_{ik} = 1$. In community detection, these π_{ik}'s indicate the *mixed community membership* for each node v_i, and they are unknown. Besides, $p(v_i | z_i = k; \boldsymbol{\phi}_i, \boldsymbol{\psi}_k, \Sigma_k)$ is a multivariate Gaussian distribution defined as follows

$$p(v_i | z_i = k; \boldsymbol{\phi}_i, \boldsymbol{\psi}_k, \Sigma_k) = \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_k, \Sigma_k) \tag{2}$$

In community embedding, the $(\boldsymbol{\psi}_k, \Sigma_k)$'s are unknown. By optimizing Eq. 1 w.r.t. π_{ik}'s and $(\boldsymbol{\psi}_k, \Sigma_k)$'s, we achieve community detection and embedding at the same time.

3.2 Node Embedding

Traditionally, node embedding focuses on preserving first- or second-order proximity. For example, to preserve first-order proximity, LINE [24] enforces two neighboring nodes to have similar embedding by minimizing

$$O_1 = -\sum_{(v_i, v_j) \in E} \log \sigma(\boldsymbol{\phi}_j^T \boldsymbol{\phi}_i) \tag{3}$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is a sigmoid function. To preserve second-order proximity, LINE and DeepWalk [20] both enforce two nodes sharing many "contexts" (i.e., neighbors within ζ hops) to have similar embedding. In this case, each node has two roles: a node for itself and a context for some other nodes. To differentiate such roles, DeepWalk introduces an extra context embedding for each node v_j as $\boldsymbol{\phi}'_j \in \mathbb{R}^d$. Denote C_i as the set of contexts for v_i.

Then, we adopt *negative sampling* [17] to define a function for measuring how well v_i generates each of its contexts $v_j \in C_i$ as

$$\Delta_{ij} = \log \sigma(\boldsymbol{\phi}'^T_j \boldsymbol{\phi}_i) + \sum_{t=1}^{m} \mathbb{E}_{v_l \sim P_n(v_l)} [\log \sigma(-\boldsymbol{\phi}'^T_l \boldsymbol{\phi}_i)] \tag{4}$$

where $v_l \sim P_n(v_l)$ denotes sampling a node $v_l \in V$ as a "negative context" of v_i according to a probability $P_n(v_l)$. We set $P_n(v_l) \propto r_l^{3/4}$ as proposed in [17], where r_l is v_l's degree. In total, there are m negative contexts. Generally, maximizing Eq. 4 enforces node v_i's embedding $\boldsymbol{\phi}_i$ to best generate its positive contexts $\boldsymbol{\phi}'_j$'s, but not its negative contexts $\boldsymbol{\phi}'_l$'s. Then, we can minimize the following objective function to preserve the second-order proximity:

$$O_2 = -\alpha \sum_{v_i \in V} \sum_{v_j \in C_i} \Delta_{ij} \tag{5}$$

where $\alpha > 0$ is a trade-off parameter.

3.3 Closing the Loop

In order to close the loop in Fig. 2, we need to enable the feedback from community detection and community embedding to node embedding. Suppose we have identified the mixed community membership π_{ik}'s and the community embedding $(\boldsymbol{\psi}_k, \Sigma_k)$'s in Sec. 3.1. Then, we can re-use Eq. 1 to enable such feedback, by seeing the node embedding $\boldsymbol{\phi}_i$'s as *unknown*. Effectively, optimizing Eq. 1 w.r.t. $\boldsymbol{\phi}_i$'s enforces the nodes $\boldsymbol{\phi}_i$'s within the same community to get closer to the corresponding community center $\boldsymbol{\psi}_k$. That is, two nodes sharing a community are likely to have similar embedding. Compared with the first- and second-order proximity, this design enforces community-aware high-order proximity on node embedding, which is useful for community detection and embedding later. For example, in Fig. 1(a), node 3 and node 10 are directly linked, but they tend to belong to two different communities according to [35]. Therefore, by only preserving first-order proximity, we may not tell their community membership's difference well. For another example, node 9 and node 10 share a number of one-hop and two-hop neighbors, but compared with node 10, node 9 tend to be closer to community led by node 1 according to [35]. Therefore, by only preserving second-order proximity, we may not tell their community membership's difference well either.

Based on the closed loop, we optimize community detection, community embedding and node embedding together. We have three types of proximity to consider for node embedding, including first-, second- and high-order proximity. In general, there are two approaches to combine different types of proximity for node embedding: (1) "concatenation", e.g., LINE first separately optimizes O_1 and O_2, then it concatenates the two resulting embedding for each node into a long vector as the final output; (2) "unification", e.g., SDNE [29] learns a single node embedding for each node to preserve both first- and second-order proximity at the same time. In this paper, to encourage the node embedding to unify multiple types of proximity, we adopt the unification approach, and leave the other approach as future work. Consequently, based on Eq. 1, we define the objective function for community detection and embedding, as well as enforcing the high-order proximity for node embedding as

$$O_3 = -\frac{\beta}{K} \sum_{i=1}^{|V|} \log \sum_{k=1}^{K} \pi_{ik} \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_k, \Sigma_k), \tag{6}$$

where $\beta \geq 0$ is a trade-off parameter. Denote $\Phi = \{\boldsymbol{\phi}_i\}$, $\Phi' = \{\boldsymbol{\phi}'_i\}$, $\Pi = \{\pi_{ik}\}$, $\Psi = \{\boldsymbol{\psi}'_k\}$ and $\Sigma = \{\Sigma_k\}$ for $i = 1, ..., |V|$ and $k = 1, ..., K$.

Then, we unify the first- and second-order proximity for node embedding. The ultimate objective function for ComE is

$$\mathcal{L}(\Phi, \Phi', \Pi, \Psi, \Sigma) = O_1(\Phi) + O_2(\Phi, \Phi') + O_3(\Phi, \Pi, \Psi, \Sigma) \quad (7)$$

Our final optimization problem becomes:

$$(\Phi^*, \Phi'^*, \Pi^*, \Psi^*, \Sigma^*) \leftarrow \underset{\forall k, diag(\Sigma_k) > \mathbf{0}}{\arg\min} \mathcal{L}(\Phi, \Phi', \Pi, \Psi, \Sigma) \quad (8)$$

where $diag(\Sigma_k)$ returns the diagonal entries of Σ_k. We particularly introduce a constraint of $diag(\Sigma_k) > \mathbf{0}$ for each $k \in \{1, ..., K\}$ to avoid the singularity issue of optimizing $\mathcal{L}$. Similar to GMM [3], there exists degenerated solutions for optimizing $\mathcal{L}$ without any constraint. That is, when a Gaussian component collapses to a single point, the $diag(\Sigma_k)$ becomes zero, which makes O_3 become negative infinity.

4 INFERENCE

Since our objective in Eq. 8 can be seen as consisted of node embedding and community embedding, we decompose the optimization into two parts, and take an iterative approach to solve it. Specifically, we consider iteratively optimizing (Π, Ψ, Σ) with a constrained minimization given (Φ, Φ'), and optimizing (Φ, Φ') with an unconstrained minimization given (Π, Ψ, Σ). Empirically, this iterative optimization algorithm converges quickly with a reasonable initialization, e.g., we initialize (Φ, Φ') by DeepWalk results in our experiments. We report the convergence in Sec. 5.4. Next we detail this iterative optimization.

Fix (Φ, Φ')**, optimize** (Π, Ψ, Σ). In this case, Eq. 8 is simplified as inferring a , with the constraints of $diag(\Sigma_k) > \mathbf{0}$ for each $k \in \{1, ..., K\}$. To solve this constrained optimization, we adopt the approach as suggested by [3], i.e., we use *expectation maximization* (EM) algorithm [9] to infer (Π, Ψ, Σ), and meet the constraint via suitable heuristics of randomly resetting $\Sigma_k > \mathbf{0}$ and $\boldsymbol{\psi}_k \in \mathbb{R}^d$ whenever a $diag(\Sigma_k)$ starts to have zero. Particularly, by EM, we can iteratively update the (Π, Ψ, Σ) by

$$\pi_{ik} = \frac{N_k}{|V|}, \quad (9)$$

$$\boldsymbol{\psi}_k = \frac{1}{N_k} \sum_{i=1}^{|V|} \gamma_{ik} \boldsymbol{\phi}_i, \quad (10)$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{|V|} \gamma_{ik} (\boldsymbol{\phi}_i - \boldsymbol{\psi}_k)(\boldsymbol{\phi}_i - \boldsymbol{\psi}_k)^T, \quad (11)$$

where $\gamma_{ik} = \frac{\pi_{ik} \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_k, \Sigma_k)}{\sum_{k'=1}^{K} \pi_{ik'} \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_{k'}, \Sigma_{k'})}$ and $N_k = \sum_{i=1}^{|V|} \gamma_{ik}$. It is worth noting that, in practice if (Φ, Φ') are initialized reasonably (e.g., by DeepWalk in our experiments), the constraints of $diag(\Sigma_k) > \mathbf{0}$ are easily satisfied, thus the inference of (Π, Ψ, Σ) can converge quickly.

Fix (Π, Ψ, Σ)**, optimize** (Φ, Φ'). In this case, Eq. 8 is simplified as an unconstrained optimization over the node embedding with three types of proximity. Due to the summation within the logarithm term of O_3, it is inconvenient to compute the gradient of $\boldsymbol{\phi}_i$. Thus, we try to minimize an upper bound of $\mathcal{L}(\Phi, \Phi' | \Phi, \Psi, \Sigma)$ instead. Specifically, we introduce

$$O_3' = -\frac{\beta}{K} \sum_{i=1}^{|V|} \sum_{k=1}^{K} \pi_{ik} \log \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_k, \Sigma_k) \quad (12)$$

Algorithm 1 Inference algorithm for ComE

Require: graph $G = (V, E)$, #(community) K, #(paths per node) γ, walk length ℓ, context size ζ, embedding dimension d, negative context size m, parameters (α, β).
Ensure: node embedding Φ, context embedding Φ', community assignment Π, community embedding (Ψ, Σ).

1: $\mathcal{P} \leftarrow$ SamplePath(G, ℓ);
2: Initialize Φ and Φ' by DeepWalk [20] with $\mathcal{P}$;
3: **for** $iter = 1 : T_1$ **do**
4: **for** $subiter = 1 : T_2$ **do**
5: Update π_{ik}, $\boldsymbol{\psi}_k$ and Σ_k by Eq. 9, Eq. 10 and Eq. 11;
6: **for** $k = 1, ..., K$ **do**
7: **if** there exists zero in $diag(\Sigma_k)$ **then**
8: Randomly reset $\Sigma_k > \mathbf{0}$ and $\boldsymbol{\psi}_k \in \mathbb{R}^d$;
9: **for all** edge $(i, j) \in E$ **do**
10: SGD on $\boldsymbol{\phi}_i$ and $\boldsymbol{\phi}_j$ by Eq. 14;
11: **for all** path $p \in \mathcal{P}$ **do**
12: **for all** v_i in path p **do**
13: SGD on $\boldsymbol{\phi}_i$ by Eq. 15;
14: SGD on its context $\boldsymbol{\phi}_j'$'s within ζ hops by Eq. 17;
15: **for all** node $v_i \in V$ **do**
16: SGD on $\boldsymbol{\phi}_i$ by Eq. 16;

It is easy to prove that $O_3'(\Phi | \Pi, \Psi, \Sigma) \geq O_3(\Phi | \Pi, \Psi, \Sigma)$ due to the following log-concavity

$$\sum_{i=1}^{|V|} \log \sum_{k=1}^{K} \pi_{ik} \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_k, \Sigma_k) \geq \sum_{i=1}^{|V|} \sum_{k=1}^{K} \log \pi_{ik} \mathcal{N}(\boldsymbol{\phi}_i | \boldsymbol{\psi}_k, \Sigma_k) \quad (13)$$

As a result, we define

$$\mathcal{L}'(\Phi, \Phi' | \Pi, \Psi, \Sigma) = O_1(\Phi) + O_2(\Phi, \Phi') + O_3'(\Phi | \Pi, \Psi, \Sigma)$$

and, thus, $\mathcal{L}'(\Phi, \Phi' | \Pi, \Psi, \Sigma) \geq \mathcal{L}(\Phi, \Phi' | \Pi, \Psi, \Sigma)$. We optimize $\mathcal{L}'(\Phi, \Phi')$ by *stochastic gradient descent* (SGD) [17]. For each $v_i \in V$, we have

$$\frac{\partial O_1}{\partial \boldsymbol{\phi}_i} = -\sum_{(i,j) \in E} \sigma(-\boldsymbol{\phi}_j^T \boldsymbol{\phi}_i) \boldsymbol{\phi}_j, \quad (14)$$

$$\frac{\partial O_2}{\partial \boldsymbol{\phi}_i} = -\alpha \sum_{v_j \in C_i} \Big[\sigma(-\boldsymbol{\phi}_j'^T \boldsymbol{\phi}_i) \boldsymbol{\phi}_j' + \sum_{t=1}^{m} \mathbb{E}_{v_l \sim P_n(v_l)} [\sigma(\boldsymbol{\phi}_l'^T \boldsymbol{\phi}_i)(-\boldsymbol{\phi}_l')] \Big], \quad (15)$$

$$\frac{\partial O_3'}{\partial \boldsymbol{\phi}_i} = \frac{\beta}{K} \sum_{k=1}^{K} \pi_{ik} \Sigma_k^{-1} (\boldsymbol{\phi}_i - \boldsymbol{\psi}_k). \quad (16)$$

We also compute the gradient for context embedding as

$$\frac{\partial O_2}{\partial \boldsymbol{\phi}_j'} = -\alpha \sum_{v_i \in V} \Big[\delta(v_j \in C_i) \sigma(-\boldsymbol{\phi}_j'^T \boldsymbol{\phi}_i) \boldsymbol{\phi}_i + \sum_{t=1}^{m} \mathbb{E}_{v_l \sim P_n(v_l)} [\delta(v_l = v_j) \sigma(\boldsymbol{\phi}_l'^T \boldsymbol{\phi}_i)(-\boldsymbol{\phi}_i)] \Big] \quad (17)$$

Algorithm and complexity. We summarize the inference algorithm of ComE in Alg. 1. In line 1, for each $v_i \in V$, we sample γ paths starting from v_i with length ℓ on G. In line 2, we initialize (Φ, Φ') by DeepWalk. In lines 4–8, we fix (Φ, Φ') and optimize (Π, Ψ, Σ) for community detection and embedding. In lines 9–16, we fix (Π, Ψ, Σ) and optimize (Φ, Φ') for node embedding.

Particularly, we update node embedding by first-order proximity (lines 9–10), second-order proximity (lines 11–14) and community-aware high-order proximity (lines 15–16). We analyze the complexity of Alg. 1. Path sampling in line 1 takes $O(|V|\gamma\ell)$. Parameter initialization by DeepWalk in line 2 takes $O(|V|)$. Community detection and embedding in line 5 takes $O(|V|K)$. Checking constraint in lines 6–8 takes $O(K)$. Node embedding w.r.t. first-order proximity in lines 9–10 takes $O(|E|)$. Node embedding w.r.t. second-order proximity in lines 11–14 takes $O(|V|\gamma\ell)$. Node embedding w.r.t. community-aware high-order proximity in lines 15–16 takes $O(|V|K)$. In total, the complexity is $O(|V|\gamma\ell+|V|+T_1\times(T_2|V|K+K+|E|+|V|\gamma\ell+|V|K))$, which is linear to the graph size (i.e., $|V|$ and $|E|$). Thus, Alg. 1 is efficient.

5 EXPERIMENTS

As community embedding is useful for visualizing communities in a low-dimensional space, as well as helping both community detection and node embedding, we design three evaluation tasks for experiments: *graph visualization*, *community detection* and *node classification*. Besides, we also empirically study the model convergence and parameter sensitivity in this section. We provide the code used during the experiments at the following link[3].

Datasets. We use five public graph datasets for evaluation. These graphs are of various types, ranging from social networks to word co-occurrence network and academic paper citation network.

• *BlogCatalog*[4] is a social network for users to publish blogs. In this dataset, each node is a BlogCatalog user and each edge is a friendship connection. Each node has multiple labels, indicating the topics of the user's blog topics.

• *Flickr*[5] is social network for users to share images and videos. In this dataset, each node is a Flickr user and each edge is a friendship connection. Each node has a label, indicating the user's interest group such as "*Sea Explorer*".

• *Wikipedia*[6] is a co-occurrence network of words appearing in the fist million bytes of the Wikipedia dump. In this dataset, each node is a word and each edge is a word co-occurrence relationship. Each node has a label, indicating the word's part-of-speech tag.

• *DBLP*[7] is an academic paper citation network built upon the DBLP repository. We extracted the papers from 19 selected conferences from five areas, as shown in Tab. 4. In this dataset, each node is a paper and each edge is a citation. Each node has a label, indicating one of the five areas for its paper's conference venue.

• *Karate Club* is a social network of a university karate club [35].

We summarize the statistics and the evaluation tasks for each dataset in Tab. 3.

Evaluation metrics. In community detection, we use both *conductance* [14] and *normalized mutual information* (NMI) [27]. Conductance is basically a ratio between the number of edges leaving a community and that within the community. NMI measures the closeness between the predicted communities with ground truth based on the node labels.

[3]https://github.com/andompesta/ComE.git
[4]http://socialcomputing.asu.edu/datasets/BlogCatalog3
[5]http://socialcomputing.asu.edu/datasets/Flickr
[6]http://snap.stanford.edu/node2vec/POS.mat
[7]https://aminer.org/billboard/aminernetwork

Table 3: Datasets used for evaluation. Task "d" denotes community detection, task "c" denotes node classification, and task "v" denotes graph visualization.

	#(node)	#(edge)	#(class)	#(label)/node	tasks
BlogCatalog	10,312	333,983	39	≥ 1	d + c
Flickr	80,513	5,899,882	195	≥ 1	d + c
Wikipedia	4,777	184,812	40	≥ 1	d + c
DBLP	13,184	48,018	5	1	v + d + c
Karate Club	34	78	2	≥ 1	v

Table 4: DBLP dataset labels.

Conference	Label
EMNLP, ACL, CoNLL, COLING	NLP
CVPR, ICCV, ICIP, SIGGRAPH	Computer Vision
KDD, ICDM, CIKM, WSDM	Data Mining
SIGMOD, VLDB/PVLDB, ICDE	Database
INFOCOM, SIGCOM, MobiHoc, MobiCom	Networking

In node classification, we use *micro-F1* and *macro-F1* [20]. Micro-F1 is the overall F1 w.r.t. all kinds of labels. Macro-F1 is the average of F1 scores w.r.t. each kind of label.

Baselines. We design baselines to back up our arguments in Sec. 1 that it is non-trivial to learn community embedding.

• *SF*: We first design a straightforward approach that separates community detection and node embedding, later fits community embedding from the detection and node embedding results. We use Spectral Clustering [25] for community detection and use DeepWalk for node embedding; finally we use GMM to fit community embedding. Note that, since its node embedding is the same as DeepWalk, we will only evaluate SF in community detection and community visualization for the Karate dataset.

Besides, we also consider the other approach that runs community detection on the node embedding results from the following state-of-the-art baselines and, then, runs GMM to detect communities and fit community embedding.

• *DeepWalk* [20]: it models second-order proximity in the embedding process.

• *LINE* [24]: it considers both first- and second-order proximity.

• *Node2Vec* [12]: it extends DeepWalk by exploiting homophily and structural roles in embedding.

• *GraRep* [5]: it models random walk based high-order proximity.

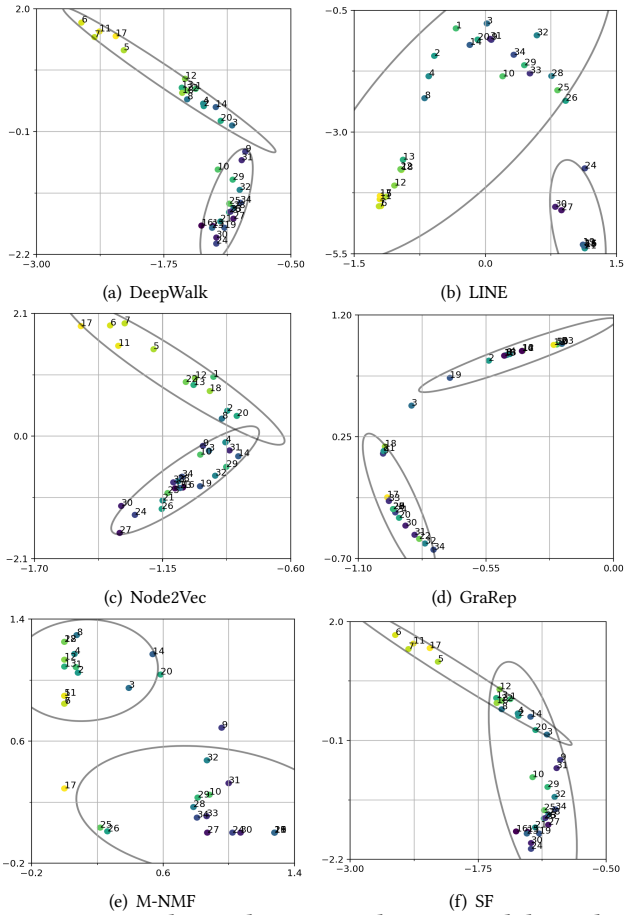
• *M-NMF* [31]: it jointly models node and community embedding using non-negative matrix factorization.

We compared our model with all the baselines on all the datasets, using the author-published codes. However, since baselines like GraRep, M-NMF often had to compute dense adjacency matrices we encounter unmanageable out-of-memory errors when running these baselines even with a 64GB-memory machine for the Flickr dataset. Similarly, Node2Vec is unfeasible on Flicker, since it has to compute and store the transition probability for each neighborhood of each node in order to perform non-uniform sampling from discrete distributions.

Parameters and environment. Our ComE has only two more parameters (i.e., α and β) than DeepWalk. To obtain a fair comparison we follow the DeepWalk and Node2Vec works to set the parameters. Unless stated otherwise, for all the methods, we set the embedding dimension $d = 128$. For DeepWalk, Node2Vec and ComE, we set $\gamma = \zeta = 10$, $\ell = 80$ and $m = 5$. As our method, also Node2Vec present two more parameters that need to be tuned. For BlogCatalog and Wikipedia, we follow the work done in [12], where $p = 0.25$ and $q = 0.25$ result as the best tuning for BlogCatalog; while Wikipedia better perform with $p = 4$ and $q = 1$. We followed the same tuning procedure also for DBLP and we found out that, like BlogCatalog, $p = 0.25$ and $q = 0.25$ works at best. For M-NMF, we tune α and β in the range $[0.1, 1, 5, 10]$ while keeping the other parameter fixed. The final setting is: (1) $\alpha = 0.1$ and $\beta = 5$ for BlogCatalog and Wikipedia; (2) while for DBLP we used $\alpha = 10$ and $\beta = 5$. For all the datasets, we set K as the number of unique labels. We run experiments on Linux machines with eight 3.50GHz Intel Xeon(R) CPUs and 16GB memory.

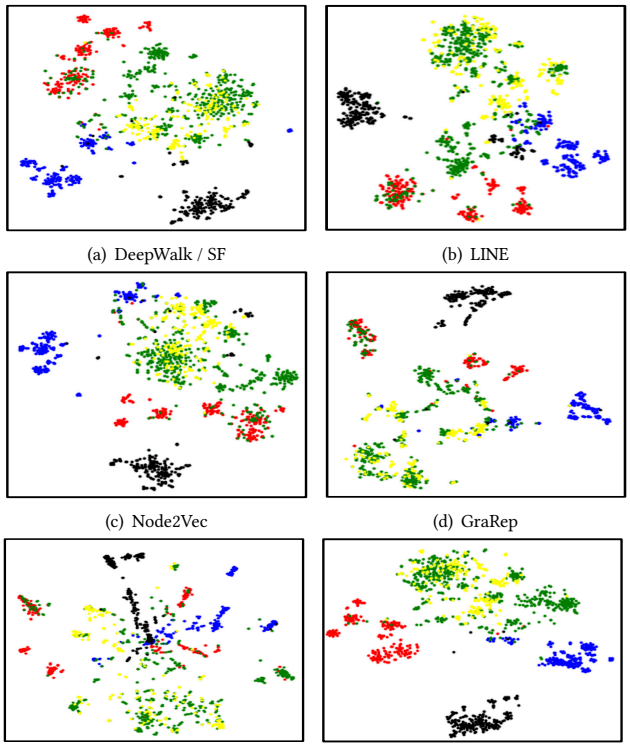
5.1 Graph Visualization

We compare ComE with the baselines on both a small Karate Club graph and a bigger DBLP paper citation graph. We visualize the Karate Club graph in Fig. 3, based on the node and community embedding results of our baselines.



(a) DeepWalk (b) LINE (c) Node2Vec (d) GraRep (e) M-NMF (f) SF

Figure 3: Graph visualization on the Karate Club graph.

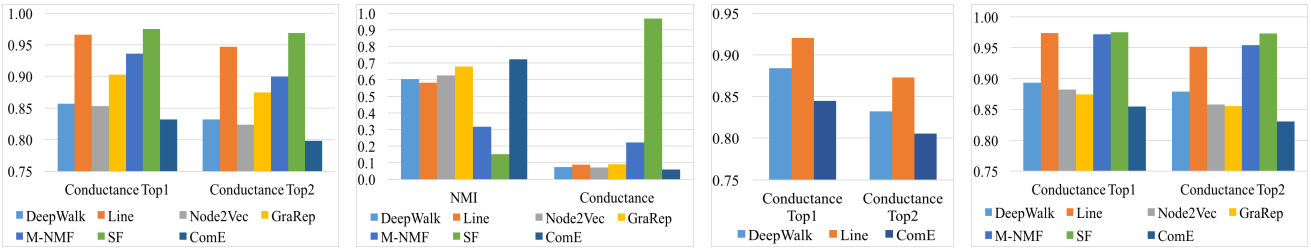


(a) DeepWalk / SF (b) LINE (c) Node2Vec (d) GraRep (e) M-NMF (f) ComE

Figure 4: Graph visualization on the DBLP graph (better viewed in color). Different node colors indicate different communities; red is *NLP*, blue is *Computer Vision*, green is *Data Mining*, yellow is *Database* and black is *Networking*

As we can see, LINE does not present community structures, since it does not consider the community in its node embedding. DeepWalk, Node2Vec, GraRep and M-NMF tend to present two separate communities, which cannot identify those weak supporters for the communities (e.g., node 9), while SF detect noise communities due to the Spectral Clustering. In contrast, as shown in Fig. 1(b), our ComE can clearly identify such weak supporters thanks to its joint modeling of overlapping community detection, community embedding and node embedding.

We visualize the DBLP paper citation graph in Fig. 4. We use the t-SNE toolkit [28] for graph visualization. Instead of plotting community embedding eclipses, in t-SNE we use different node colors to visualize different communities and see if the algorithms are able to correctly preserve the communities present in the graphs. Note that, although SF and DeepWalk have different way to generate the community embedding, they use the same node embedding, thus, they have the visualization results in Fig. 4(a). First of all, note that no method is able to separate the green and the yellow classes (Data Mining and Database), we believe that this is related to the dataset itself given the similarity of the two topics. Moreover, DeepWalk and Node2Vec generate representations with many overlaps among the colors and the overall embedding is quite similar. This could be possible because both the methodology model only the second-order proximity.



(a) BlogCatalog (b) DBLP (c) Flickr (d) Wikipedia

Figure 5: Community detection results. The smaller conductance is, the better. The bigger NMI is, the better.

LINE presents a bit more cohesive visualization; probably this is due to its ability in preserving first and second order proximity. We also observe that, in GraRep's results, green nodes are spread in different places and mixed with red and yellow nodes, meaning a lack of clear community structure in the embedding. This might be possible due to its ability in decomposing a high-order transition probability matrix, which could be dominate by Data Mining and Database nodes. On the other hand, it is able to correctly detect Networking and Computer vision community. M-NMF, instead, present blurred community structure. This could be related to the fact that M-NMF relay on the modularity matrix to learn the community embedding, which could be noise due to the overlap between the green and yellow nodes. This hypothesis is also supported by the karate results (Fig. 3(e)), in which M-NMF already demonstrate to struggle in modeling overlapping communities. Compared with all these baseline methods, our ComE can correctly detect three classes thanks to its joint modeling, but as for the other methods is not able to differentiate between green and yellow nodes which are clustered in mixed communities.

5.2 Community Detection

In community detection, our goal is to predict the most likely community assignment for each node. As an unsupervised task, we use the whole graph for learning embeddings and, hence, predicting communities for each node. Note that in BlogCatalog, Wikipedia and Flickr are multi-labels datasets, so we compute only the conductance for the Top 2 communities of each node since there is no clear way to calculate NMI for such a multi-label setting.

During those experiments we set $\alpha = 0.1, \beta = 0.1$ in ComE for all the datasets. As shown in Fig. 5, ComE is consistently better than the baselines in terms of both conductance and NMI. For the conductance ComE improves the best baselines by relatively 2.2% to 3.1% on BlogCatalog and Wikipedia, 4.4% and 3.2% in Flickr, and 17.1% in DBLP. Under NMI metrics ComE achieve an improvement of 6.7%. These improvements suggest that, modeling community detection together with node embedding is better than solving them separately. The general poor performance of SF methods also support the hypothesis of having the closed loop model. Besides, we also observe that, on average, the graph embedding methods perform better than the others, which suggest the usefulness of consider graph embedding for community detection. In particular, Node2Vec happen to be the best baseline for both DBLP and BlogCatalog datasets, while in Wikipedia GraRep outperform all the others. Finally, we observe that, accordingly to the findings in Sec. 5.1, M-NMF struggle in modeling multi-label datasets.

5.3 Node Classification

In node classification, our goal is to categorize each node into one or more classes, depending on whether it is a single-label or multi-label setting. We follow [20] to first train graph embedding on the whole graph, then randomly split 10% (BlogCatalog, Wikipedia and DBLP) and 90% (Flickr) of nodes as test data, respectively. We use the remaining nodes, together with their labels, to train a classifier by LibSVM ($c = 1$ for all the methods) [6]. We repeat 10 times and report the average results.

We compare ComE with all the baselines in terms of node classification. We set $\alpha = 0.1$, $\beta = 0.01$ for all the datasets except for DBLP where we kept $\alpha = 0.1$ and $\beta = 0.1$. We vary the number of training data to build the classifiers for each method. As shown in Tab. 5, ComE is generally better than the baselines in terms of both macro-F1 and micro-F1. In particular, ComE improves the best baselines by relatively 0.8% to 22.6% (macro-F1) and 0.71% to 48% (micro-F1), when using 80% (BlogCatalog, Wikipedia and DBLP) and 26.9% (macro) 1.5% (micro) when using 8% (Flickr) of labeled nodes for training. Our student t-tests show that all the above relative improvements are significant over the 10 data splits, with one-tailed p-values always less than 0.01. It is interesting to see ComE improves the baselines on node classification, since it is unsupervised and it does not directly optimize the classification loss. This implies the high-order proximity from community embedding does contribute to node embedding. In addition, we also make some interesting observations from Tab. 5.

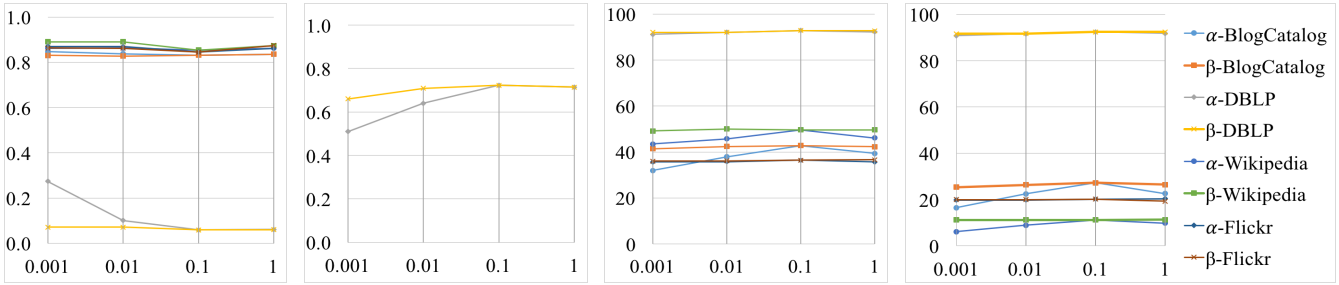
Firstly, in Wikipedia, GraRep is better than our ComE when using less than 50% labeled nodes in training under the Micro-F1 score. A possible reason is that, Wikipedia contains a much smaller number of nodes than the rest of the datasets, leading to a comparatively smaller set of sampled paths. On the other hand, GraRep used the transition probability matrix, which could contain more information than the sample path, to learn higher-order of proximity. This provides supplemental information to train the classifiers with limited training labels. However, as more labeled data is available, this advantage of GraRep becomes smaller, and our ComE starts to outperform.

Secondly, it is possible to notice how in BlogCatalog, GraRep is the best baseline; meanwhile Node2Vec outperforms the other baselines in DBLP. This maybe because the number of edges in DBLP is much fewer than those of the other datasets. In a graph with low average degree, it is easier for Node2Vec learning which are the good neighborhood to explore. Moreover, respect tho all the other datasets, in Dblp the random walk based methods works relatively better than the factorization based methods.

Table 5: Node classification results.

		BlogCatalog									Flickr								
	% Labels	10%	20%	30%	40%	50%	60%	70%	80%	90%	1%	2%	3%	4%	5%	6%	7%	8%	9%
Macro-F1 (%)	ComE	19.3	**22.4**	**23.5**	**24.8**	**25.1**	**25.4**	**25.7**	**26.2**	**26.4**	**3.5**	**5.3**	**10.1**	**15.4**	**18.2**	**19.0**	**19.3**	**19.7**	**20.0**
	DeepWalk/SF	17.2	18.9	19.9	20.6	20.9	21.4	21.5	21.5	21.5	1.9	3.1	7.9	11.6	13.8	14.3	15.2	15.5	15.7
	Line	7.6	8.6	9.5	10.0	10.2	10.8	10.8	10.9	10.9	1.6	2.5	6.0	8.7	10.2	10.5	11.0	11.2	11.2
	Node2Vec	18.3	20.5	21.9	22.8	23.1	23.4	23.6	23.6	23.6	-	-	-	-	-	-	-	-	-
	GraRep	**19.6**	21.0	22.3	22.8	22.9	23.3	23.5	23.6	24.0	-	-	-	-	-	-	-	-	-
	M-NMF	12.8	13.7	14.7	15.2	15.3	15.3	15.6	15.7	15.8	-	-	-	-	-	-	-	-	-
Micro-F1 (%)	ComE (ours)	30.1	**35.9**	**37.3**	**39.6**	**40.3**	**40.7**	**41.0**	**41.4**	**42.4**	**10.2**	**12.2**	**18.7**	**29.1**	**34.6**	**35.1**	**35.5**	**35.9**	**36.1**
	DeepWalk/SF	27.1	29.8	33.2	35.4	35.9	37.9	38.0	38.3	39.0	6.1	7.9	17.4	27.1	32.9	33.2	35.1	35.3	35.5
	Line	18.5	21.5	25.2	27.2	27.6	29.5	29.8	30.2	30.5	3.9	5.5	14.7	24.5	30.3	30.6	32.3	32.5	32.6
	Node2Vec	27.8	31.0	34.7	36.9	37.4	39.4	39.6	39.9	40.5	-	-	-	-	-	-	-	-	
	GraRep	**30.4**	32.9	36.3	38.2	38.6	40.2	40.7	40.9	42.0	-	-	-	-	-	-	-	-	
	M-NMF	23.2	25.7	29.0	31.0	31.5	33.2	33.4	33.8	34.4	-	-	-	-	-	-	-	-	-

		Wikipedia									DBLP								
	% Labels	10%	20%	30%	40%	50%	60%	70%	80%	90%	10%	20%	30%	40%	50%	60%	70%	80%	90%
Macro-F1 (%)	ComE	**5.5**	**5.5**	**4.9**	**4.9**	**5.6**	**6.9**	**8.6**	**10.6**	**11.2**	**91.1**	**91.6**	**91.8**	**92.0**	**92.1**	**92.2**	**92.2**	**92.2**	**92.4**
	DeepWalk/SF	2.3	2.4	2.6	2.7	3.9	4.4	5.1	6.2	10.9	89.7	90.5	90.8	91.0	91.1	91.2	91.2	91.2	91.4
	Line	2.1	2.3	2.4	2.5	2.8	3.3	3.9	5.3	8.9	89.2	89.8	90.1	90.2	90.3	90.3	90.4	90.4	90.7
	Node2Vec	2.9	2.9	3.1	3.3	4.1	5.8	7.7	8.3	9.5	90.0	90.7	91.0	91.2	91.4	91.4	91.5	91.5	91.7
	GraRep	4.5	4.8	4.8	4.8	4.8	5.6	6.3	8.1	11.0	89.9	90.1	90.3	90.4	90.5	90.5	90.5	90.6	90.8
	M-NMF	1.7	1.7	1.7	1.7	1.8	2.3	3.0	3.9	5.3	88.3	88.8	89.2	89.3	89.4	89.5	89.5	89.6	89.8
Micro-F1 (%)	ComE	**25.4**	25.3	24.6	24.3	**27.7**	**31.2**	**38.3**	**49.5**	**50.0**	**91.6**	**92.0**	**92.2**	**92.4**	**92.5**	**92.6**	**92.6**	**92.6**	**92.8**
	DeepWalk/SF	20.0	20.2	20.4	20.9	24.1	25.1	26.8	31.1	45.3	90.3	90.9	91.2	91.4	91.6	91.6	91.6	91.6	91.8
	Line	21.5	22.2	22.4	22.5	23.0	24.4	26.1	30.5	44.2	89.9	90.5	90.7	90.8	90.9	91.0	91.0	91.0	91.4
	Node2Vec	20.7	21.2	21.3	21.4	25.6	29.7	37.4	40.6	47.3	90.6	91.2	91.5	91.7	91.9	91.9	91.9	92.0	92.2
	GraRep	24.6	**25.4**	**25.8**	**25.8**	26.1	27.1	29.1	33.4	46.0	90.4	90.6	90.8	90.9	91.0	91.1	91.1	91.1	91.4
	M-NMF	19.6	20.4	21.2	21.6	22.5	23.9	25.7	30.5	45.1	89.1	89.5	89.9	90.0	90.1	90.2	90.2	90.3	90.5



(a) Conductance (b) NMI (c) Micro-F1 (d) Macro-F1

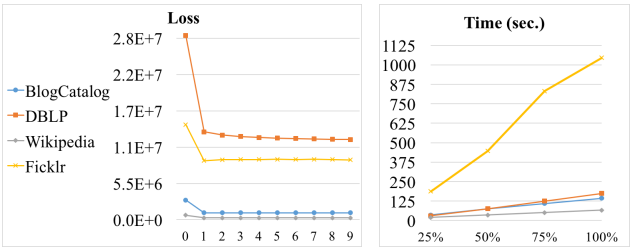
Figure 6: Impact of the parameters α and β. Note that NMI has only DBLP since is the only single-label dataset.

This suggests that the average degree of the graph can impact the performance due to the length and the windows size limitation of the random walk methods. Besides this intuition, a deeper investigation is needed to better understand the limitation and the advantages of the random walks methods.

Thirdly, it is interestingly to notice how in node classification the best baseline is the opposite respect community detection. In Sec. 5.2 Node2Vec was the best baseline for BlogCatalog and DBLP, while in node classification task GraRep outperform Node2Vec. This suggest that focusing on community level as higher order of proximity is more valuable than learning the graph structure form the transition probability matrix, since lead to better performances in a wider tasks set.

5.4 Model Study

We tune the model parameters α (trade-off parameter for context embedding) and β (trade-off parameter for community embedding) in Fig. 6. In each figure, we tune one parameters α and β within the range of $[0.001, 1]$ while fixing the other parameter as 0.1. In general, the model performance is quite robust when α and β are within the range of $[0.001, 1]$. Specifically, $\alpha = 0.1$ gives the best trade off for the second-order proximity in the objective function as it provides the best community prediction results in Wikipedia and DBLP, and the best node classification results in BlogCatalog and Flickr. The tuning of β is not as sensitive as α, especially in terms of node classification task. However, as indicated by community detection performance in BlogCatalog, $\beta = 0.1$ is the best trade-off for community embedding.



(a) Convergence (b) Efficiency

Figure 7: Model's convergence and efficiency

We further validate the convergence and efficiency of ComE in Fig. 7. We record the value of the loss function (Eq. 7) at the end of every iteration (line 18 in Alg. 1) to show the convergence of our proposed ComE. Note that we normalize the loss value by $|V|$ for different datasets so as to better illustrate them in one figure. As shown in Fig. 7(a), the loss of ComE converges quickly within 2–3 iterations. To demonstrate the efficiency of ComE, we test it on all the four datasets at different scales.

For each dataset, we generate three subsets in which we keep 25%, 50% and 75% of the total number of edges and nodes, respectively. Note that, to speed up the computation time of those experiments we set $d = 2$ and $\zeta = 5$. The diagram in Fig. 7(b) shows the processing time of ComE (line 2 – line 19 in Alg. 1) in different datasets. Clearly, the processing time of ComE is linear to the graph size (i.e., $|V|$ and $|E|$). This validates our complexity analysis at the end of Sec. 4.

6 CONCLUSION

In this paper, we studied the important, yet largely under-explored problem of embedding communities on graphs. We developed a closed loop among community embedding, community detection and node embedding. Therefore, we tried to jointly optimize all these three tasks, in order to allow them to reinforce each other.

We then developed a scalable inference algorithm, which only requires a complexity of $O(|V| + |E|)$. We evaluated our model on multiple real-world datasets and showed that our model outperforms the state-of-the-art baselines by at least 6.6% (NMI) and 2.2%–16.9% (conductance) in community detection, 0.8%–26.9% (macro-F1) and 0.71%–48% (micro-F1) in node classification. It also improved the baselines in the task of graph visualization.

ACKNOWLEDGEMENTS

We thank the support of: National Natural Science Foundation of China (No. 61502418), Research Grant for Human-centered Cyber-physical Systems Programme at Advanced Digital Sciences Center from Singapore A*STAR, National Science Foundation IIS 16-19302 and CSD-Centro Sistemi Direzionali.

REFERENCES

[1] Shiyu Chang andRSM:Dai2016 Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C. Aggarwal, and Thomas S. Huang. 2015. Heterogeneous Network Embedding via Deep Architectures. In *KDD*. 119–128.
[2] Mikhail Belkin and Partha Niyogi. 2001. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *NIPS*. 585–591.
[3] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., NJ, USA.
[4] HongYun Cai, Vincent W. Zheng, Fanwei Zhu, Kevin Chen-Chuan Chang, and Zi Huang. 2017. From Community Detection to Community Profiling. *PVLDB* 10, 7 (2017), 817–828.
[5] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning Graph Representations with Global Structural Information. In *CIKM*. 891–900.
[6] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.* 2, 3 (May 2011), 27:1–27:27.
[7] Trevor F. Cox and M.A.A. Cox. 2000. *Multidimensional Scaling, Second Edition* (2 ed.). Chapman and Hall/CRC.
[8] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In *ICML*. 2702–2711.
[9] Arther P. Dempster, Nan M. Laird, and Donald B. Rubin. 1977. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* 39, 1 (1977), 1–38.
[10] Hanyin Fang, Fei Wu, Zhou Zhao, Xinyu Duan, Yueting Zhuang, and Martin Ester. 2016. Community-Based Question Answering via Heterogeneous Social Network Learning. In *AAAI*. 122–128.
[11] Yuan Fang, Wenqing Lin, Vincent W. Zheng, Min Wu, Kevin Chen-Chuan Chang, and Xiaoli Li. 2016. Semantic proximity search on graphs with metagraph-based learning. In *ICDE*. 277–288.
[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*.
[13] X. He, D. Cai, Y. Shao, H. Bao, and J. Han. 2011. Laplacian Regularized Gaussian Mixture Model for Data Clustering. *TKDE* 23, 9 (2011), 1406–1418.
[14] Kyle Kloster and David F. Gleich. 2014. Heat Kernel Based Community Detection. In *KDD*. 1386–1395.
[15] Mark Kozdoba and Shie Mannor. 2015. Community Detection via Measure Space Embedding. In *NIPS*. 2890–2898.
[16] Zemin Liu, Vincent W. Zheng, Zhou Zhao, Fanwei Zhu, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. 2017. Semantic Proximity Search on Heterogeneous Graph by Proximity Embedding. In *AAAI*. 154–160.
[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*. 3111–3119.
[18] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *ICML*. 2014–2023.
[19] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric Transitivity Preserving Graph Embedding. In *KDD*. 1105–1114.
[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *KDD*. 701–710.
[21] Sam T. Roweis and Lawrence K. Saul. 2000. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science* 290, 5500 (2000), 2323–2326.
[22] Mrinmaya Sachan, Avinava Dubey, Shashank Srivastava, Eric P. Xing, and Eduard Hovy. 2014. Spatial Compactness Meets Topical Consistency: Jointly Modeling Links and Content for Community Detection. In *WSDM*. 503–512.
[23] Yizhou Sun, Charu C. Aggarwal, and Jiawei Han. 2012. Relation Strength-aware Clustering of Heterogeneous Information Networks with Incomplete Attributes. *PVLDB* 5, 5 (Jan. 2012), 394–405.
[24] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *WWW*. 1067–1077.
[25] Lei Tang and Huan Liu. 2011. Leveraging social media networks for classification. *Data Min. Knowl. Discov.* 23, 3 (2011), 447–478.
[26] 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.
[27] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. 2014. Learning Deep Representations for Graph Clustering. In *AAAI*. 1293–1299.
[28] L.J.P. van der Maaten and G.E. Hinton. 2008. Visualizing High-Dimensional Data Using t-SNE. *JMLR* 9 (2008), 2579–2605.
[29] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural Deep Network Embedding. In *KDD*. 1225–1234.
[30] Meng Wang, Chaokun Wang, Jeffrey Xu Yu, and Jun Zhang. 2015. Community Detection in Social Networks: An In-depth Benchmarking Study with a Procedure-oriented Framework. *PVLDB* 8, 10 (June 2015), 998–1009.
[31] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding. In *AAAI*. 203–209.
[32] Jierui Xie, Stephen Kelley, and Boleslaw K. Szymanski. 2013. Overlapping Community Detection in Networks: The State-of-the-art and Comparative Study. *ACM CSUR* 45, 4 (2013), 43:1–43:35.
[33] Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. 2016. Representation Learning of Knowledge Graphs with Entity Descriptions. In *AAAI*. 2659–2665.
[34] Liang Yang, Xiaochun Cao, Dongxiao He, Chuan Wang, Xiao Wang, and Weixiong Zhang. 2016. Modularity Based Community Detection with Deep Learning. In *IJCAI*. 2252–2258.
[35] Wayne W. Zachary. 1977. An Information Flow Model for Conflict and Fission in Small Groups. *Journal of Anthropological Research* 33, 4 (1977), 452–473.