Hierarchical Taxonomy Aware Network Embedding

Jianxin Ma[*]
Tsinghua University
majx13fromthu@gmail.com

Peng Cui
Tsinghua University
cuip@tsinghua.edu.cn

Xiao Wang
Tsinghua University
wangxiao007@mail.tsinghua.edu.cn

Wenwu Zhu
Tsinghua University
wwzhu@tsinghua.edu.cn

ABSTRACT

Network embedding learns the low-dimensional representations for vertices, while preserving the inter-vertex similarity reflected by the network structure. The neighborhood structure of a vertex is usually closely related with an underlying hierarchical taxonomy—the vertices are associated with successively broader categories that can be organized hierarchically. The categories of different levels reflects similarity of different granularity. The hierarchy of the taxonomy therefore requires that the learned representations support multiple levels of granularity. Moreover, the hierarchical taxonomy enables the information to flow between vertices via their common categories, and thus provides an effective mechanism for alleviating data scarcity. However, incorporating the hierarchical taxonomy into network embedding poses a great challenge (since the taxonomy is generally unknown), and it is neglected by the existing approaches. In this paper, we propose NetHiex, a NETwork embedding model that captures the latent HIErarchical taXonomy. In our model, a vertex representation consists of multiple components that are associated with categories of different granularity. The representations of both the vertices and the categories are co-regularized. We employ the nested Chinese restaurant process to guide the search of the most plausible hierarchical taxonomy. The network structure is then recovered from the latent representations via a Bernoulli distribution. The whole model is unified within a nonparametric probabilistic framework. A scalable expectation-maximization algorithm is derived for optimization. Empirical results demonstrate that NetHiex achieves significant performance gain over the state-of-arts.

KEYWORDS

Network embedding; Network representation learning; Hierarchical taxonomy; Nested Chinese restaurant process

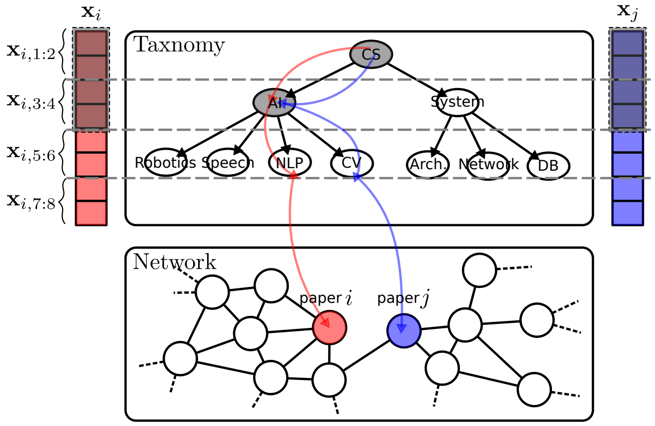


Figure 1: The hierarchical taxonomy is a hierarchy of categories. Each vertex is assigned a path (from the root to a leaf) in the hierarchy. Each category in the taxonomy has its own representation. Our model regularizes the representation of a vertex so that it centers around its associated category representations. In the example, the height of the hierarchy is $L = 3$. $\mathbf{x}_i \in \mathbb{R}^8$ is the representation of vertex i. $\mathbf{x}_{i,1:2}$, $\mathbf{x}_{i,3:4}$, and $\mathbf{x}_{i,5:6}$ should be close to the representations of category CS, AI, and NLP, respectively, while $\mathbf{x}_{i,7:8}$ is for characterizing individual differences and not associated with any category.

ACM Reference Format:
Jianxin Ma, Peng Cui, Xiao Wang, and Wenwu Zhu. 2018. Hierarchical Taxonomy Aware Network Embedding. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220062

1 INTRODUCTION

Network embedding, also known as network representation learning, is a recently proposed paradigm that automates the process of extracting continuous feature vectors for vertices in a network. One basic requirement of network embedding is that the learned vertex representations should preserve the inter-vertex similarity reflected by the network structure [6, 11, 14]. Therefore various network embedding methods have been proposed to preserve the first-, second-, and kth-order proximity [7, 21, 29], and the neighborhood structure explored by random walks [13, 23], etc.

The neighborhood structure is usually closely related with an underlying hierarchical taxonomy, where the successively broader categories associated with the vertices are hierarchically organized and form the tree hierarchy of the taxonomy. The categories of different levels reveals similarity of different granularity. For instance,

[*] Beijing National Research Center for Information Science and Technology (BNRist)

KDD '18, August 19–23, 2018, London, United Kingdom
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220062

in the citation network shown in Figure 1, a paper on Natural Language Processing (NLP) and a paper on Computer Vision (CV) belong to two different categories and are deemed dissimilar according to their fine-grained neighborhood structure. Nonetheless, the paper on NLP and the paper on CV can actually be considered similar when looking at the coarser-grained structure, since they both belong to the Artificial Intelligence (AI) category. The hierarchy of the taxonomy thus requires that the vertex representations encode the structural information across mulitple levels of granularity, so as to well support the wide variety of downstream applications.

Moreover, the hierarchical taxonomy provides an effective mechanism for alleviating the data scarcity issue. Real world networks are usually extremely sparse, due to either the difficulty of collecting comprehensive data, or the fact that the entities are not exposed enough to each other for most links to form. The hierarchical taxonomy can facilitate extracting, storing, and reusing the common knowledge associated with each category. Consequently, the relations between two remotely connected vertices (that are several hops away from each other) will be strengthened if the two vertices share common categories in the hierarchy.

However, the hierarchical taxonomy has been largely neglected by the existing network embedding approaches, and incorporating it into network embedding poses great challenges. We are faced with the daunting task of searching for the proper taxonomy in the combinatorial space of countless tree structures, because the underlying hierarchical taxonomy is generally unknown.

In this paper, we propose NetHiex, a network embedding model that simultaneously detects and leverages the underlying hierarchical taxonomy. As illustrated in Figure 1, a vertex representation in our model consists of multiple components, each of which corresponds to a category at the different layer of the hierarchy [1]. The categories of different granularity associated with the vertex then form a path from the root to a leaf in the tree hierarchy. We employ the nested Chinese restaurant process (nCRP) [3, 4] as a prior distribution of the tree structure and the paths. The latent representations and the observed network structure are then connected with a Bernoulli distribution. Intuitively, the categories at the top of the hierarchy will be passed by more paths than the categories at the bottom, and therefore tend to represent coarser-grained categories. As a result, the different components of vertex representations that are regularized by the categories at the different levels will lean towards capturing features of different granularity. Moreover, the categories distill common knowledge from related vertex representations, and store it in the form of category representations. The relation between two remotely connected vertices is thus strengthened, because they now share information via the common categories passed by their paths. Overall, our model is unified within a nonparametric probabilistic framework. We derive an efficient expectation-maximization (EM) algorithm to estimate the parameters. The overall time complexity of our algorithm per iteration is linear in the size of the network, making our approach suitable for real world large-scale networks.

We empirically assess our approach on several benchmark datasets. The results show that our approach can significantly outperform the state-of-the-art approaches in various tasks, such as classification and link prediction, and is robust to missing data. We further examine the hierarchical taxonomy learned on a word co-occurrence network, and visualize the multiple levels of granularity.

The contributions of our paper are summarized as follows:

- We study the important problem of incorporating the hierarchical taxonomy into network embedding. Our approach, NetHiex, is able to learn representations that preserve both the fine-grained and the coarse-grained network structure.
- We derive an efficient EM algorithm, each iteration of which has a linear time complexity. This makes our approach highly competitive with the existing scalable ones.
- Extensive quantitative and qualitative experiments demonstrate the merits of learning representations with multiple levels of granularity and alleviating data scarcity with the hierarchical taxonomy.

2 THE HIERARCHICAL MODEL

In this section, we present NetHiex, a unified probabilistic framework for network embedding with a latent hierarchical taxonomy.

2.1 Model Description

We use a tree of height L (all the paths from the root to a leaf is of length L) to represent the hierarchy of categories (e.g. Figure 1). Given a network with N vertices, we assume that each vertex is associated with a path (from the root to leaf) in the tree hierarchy. Let $\mathbf{c}_n$ be the path of vertex n. The path $\mathbf{c}_n$ indicates a series of successively finer-grained categories to which vertex n belongs.

The exact structure of the tree hierarchy is unknown, and neither do we know to which path each vertex is assigned. We therefore use the nested Chinese restaurant process (nCRP) [4] as the prior distribution over the tree structure and the paths. In other words, the nCRP provides the prior probability

$$p(\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_N).$$

We will give the detailed definition of the nCRP in the next subsection. For the moment, let us focus on the rest part of the model, so as to provide a comprehensive picture of our approach.

NetHiex aims to learn d-dimensional representations of the vertices, i.e. $\{\mathbf{x}_n\}_{n=1}^N \subset \mathbb{R}^d$, as well as the representations of the categories in the tree hierarchy, i.e. $\{\mathbf{w}_t : \text{category } t \text{ in the hierarchy}\} \subset \mathbb{R}^{\Delta d}$, where $\Delta d = \left\lfloor \frac{d}{L+1} \right\rfloor$.

The prior distribution of the category representations is

$$\mathbf{w}_t \sim \text{Normal}\left(\mathbf{0}, \sigma_w^2 \mathbf{I}\right).$$

We set $\sigma_w \to \infty$ in our implementation, since we would like the categories to be as diverse as possible. In other words, we do not regularize $\mathbf{w}_t$ during optimization.

For each vertex n, we divide $\mathbf{x}_n$ into $L + 1$ parts. Each part is of dimension Δd, except the last part, whose dimension is $(d - L\Delta d)$. The first L parts, collectively referred to as $\mathbf{x}_{n,1:L\Delta d}$, follow the normal distribution,

$$\mathbf{x}_{n,1:L\Delta d} \sim \text{Normal}\left(\mathbf{w}_{\mathbf{c}_n}, \sigma_x^2 \mathbf{I}\right),$$

[1] We assume that all the paths (from the root to a leaf) in the hierarchy to be of the same length L. Otherwise we can expand the shorter paths until they are of length L.

where σ_x is a hyper-parameter, and $\mathbf{w}_{\mathbf{c}_n}$ is the concatenation of all the $\mathbf{w}_t$ visited by the path $\mathbf{c}_n$. We then let the $(L+1)$th part of $\mathbf{x}_n$ follow Normal$(0, \infty)$, i.e. we do not regularize the last part.

Let $\mathcal{D} = \{r_{uv}\}$ be the set of the observed links. For simplicity, we assume r_{uv} to be binary, i.e. $r_{uv} = 1$ if we observe a link between vertices u and v, and $r_{uv} = 0$ if we are certain that there is no link between u and v. Our model assumes that

$$r_{uv} \sim \text{Bernoulli}\left(e^{-\frac{\|\mathbf{x}_u - \mathbf{x}_v\|^2}{l^2}}\right),$$

where $\|\cdot\|$ is the L2-norm, and l is a hyper-parameter.

In practice, we typically observe only $r_{uv} = 1$, and do not observe $r_{uv} = 0$. We therefore follow previous work [13, 29] and use negative sampling to generate a number of $r_{uv} = 0$ equal to the number of the positive observations, so as to balance the dataset. In addition, we follow LINE [29] and SDNE [36], and augment the dataset by adding a number of $r_{uv} = 1$ (equal to the number of the observed links) for vertices u and v who share common neighbors, in order to better capture second-order proximity.

2.2 The nCRP Prior

Before we can give the definition of the nCRP [4], we need to review its building block, the Chinese restaurant process (CRP) [1]. The CRP is typically used as a prior for partitioning a set of samples. Imagine a Chinese restaurant with an infinite number of tables, each with infinite capacity. The first customer sits at the first table, i.e. $t_1 = 1$. The probability of the nth customer sitting at table t_n is:

$$p\left(t_n = k \mid t_{1:(n-1)}\right) \propto \begin{cases} m_k & \text{if } k \text{ already exists,} \\ \gamma & \text{if } k \text{ is a new table,} \end{cases}$$

where m_k is the number of customers already sitting at the kth table, and γ is a hyper-parameter. Intuitively it describes the rich-get-richer phenomenon.

A nCRP with L levels organizes an infinite number of CRPs into a tree hierarchy of height L, where each non-leaf node in the tree represents a CRP (Figure 2). For example, we can think of the hierarchy as home→city→restaurant→table when $L = 4$. In this example, there are an infinite number of cities, each city with infinite number of restaurants, each restaurant with infinite number of tables. Customers decide which city to visit according to a top-level CRP represented by the root. Each customer then chooses a restaurant according to the CRP associated with the chosen city (represented by the nodes at the second layer), and chooses a table according the CRP of the chosen restaurant.

NetHiex assumes that the vertices follow an nCRP with L levels. The path $\mathbf{c}_n$ of vertex n is therefore decided according to the probability induced by the CRPs visited by the path, denoted by $p(\mathbf{c}_n \mid \mathbf{c}_{1:(n-1)})$. In other words, in our model,

$$\mathbf{c}_n \mid \mathbf{c}_{1:(n-1)} \sim \text{nCRP}\left(\gamma, \mathbf{c}_{1:(n-1)}\right).$$

The overall likelihood function of our model is therefore

$$\prod_{t \in \text{nCRP}} p(\mathbf{w}_t) \times \prod_{n=1}^{N} \left[p(\mathbf{c}_n \mid \mathbf{c}_{1:(n-1)}) p(\mathbf{x}_n \mid \mathbf{w}_{\mathbf{c}_n})\right] \times \prod_{r_{uv} \in \mathcal{D}} p(r_{uv} \mid \mathbf{x}_u, \mathbf{x}_v).$$

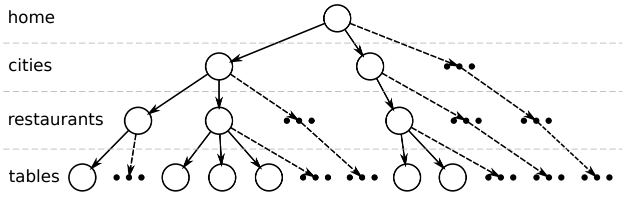


Figure 2: A nCRP with $L = 4$ levels. Each node represents a category. A non-leaf node in the nCRP represents a CRP.

2.3 Model Variants

We have mainly focused on networks that are simple graphs with no attributes. However, our model can be easily extended to handle weighted/attributed/directed networks.

In section 2.1, we use a Bernoulli distribution (that depends on $\mathbf{x}_u$ and $\mathbf{x}_v$) as the distribution over $r_{uv} \in \{0, 1\}$. If the links are weighted, we can simply replace the distribution with a Poisson distribution, a normal distribution, or a Gamma distribution, etc., depending on the type of the links. To incorporate vertex attributes (vertex features), we can modify the distribution over r_{uv} so that it depends on the attributes (in addition to $\mathbf{x}_u$ and $\mathbf{x}_v$).

To support directed networks, we can follow HOPE [21] and learn two representations ($\mathbf{x}_u \in \mathbb{R}^d$ and $\mathbf{y}_u \in \mathbb{R}^d$) for each vertex u, so as to preserve asymmetric transitivity [21]. For example, we can let $\mathbf{y}_u \sim \text{Normal}\left(\mathbf{0}, \sigma_y^2 \mathbf{I}\right)$ and $r_{uv} \sim \text{Bernoulli}(\sigma(\mathbf{x}_u^\top \mathbf{y}_v))$, where $\sigma(\cdot)$ is the sigmoid function. r_{uv} and r_{vu} will be different this way, and hence the direction of a link is preserved.

These modifications bring no technical difficulty when it comes to optimization. In fact, we only need to adjust a single sub-problem (the one involving r_{uv}) of the M-step, which is straightforward.

3 OPTIMIZATION

The likelihood function presented in the previous section is not easy to optimize, as $\mathbf{c}_n$ heavily depends on $\mathbf{c}_{1:(n-1)}$, and there is an infinite number of categories in the nCRP. We therefore use the nested stick-breaking construction [27] of the nCRP to decouple the paths, and derive an EM algorithm that optimizes a finite subtree of the nCRP at any given time and progressively expand the subtree when appropriate.

3.1 The Stick-Breaking Construction

A CRP can be reformulated as a stick-breaking process [27]. In the stick-breaking construction, each table i is associated with a latent variable v_i that follows a Beta distribution parameterized by γ, i.e. $v_i \sim \text{Beta}(1, \gamma)$. Each customer then independently chooses the ith table with probability $\pi_i = v_i \prod_{j=1}^{i-1}(1 - v_j)$.

Similarly, an nCRP can be reformulated using a nested stick-breaking construction, by replacing each CRP with a stick-breaking process. We will use the new construction hereafter, as it decouples the paths of the vertices and is much easier to optimize.

3.2 Expectation-Maximization

We propose to optimize the model with an EM algorithm. Our algorithm is based on the idea of maintaining a truncated tree [35].

A truncated tree $\mathcal{T}$ is a subtree of the infinite tree specified by the nCRP. Every path from the root to a leaf in the truncated tree is of exactly length L, as in the infinite tree. The basic idea is to fix the values of v_t and $\mathbf{w}_t$ for all the nodes outside the truncated tree to the means of their prior distributions ($p(v_t)$ and $p(\mathbf{w}_t)$), while allowing the v_t and $\mathbf{w}_t$ of the nodes inside the truncated tree to be optimized. We can then gradually increase the size of the truncated tree, since a larger truncated tree will always lead to a solution that is at least as good as the old one.

This subsection focuses on the optimization problem when the truncated tree $\mathcal{T}$ is specified and fixed. Let $V = \{v_t : t \in \mathcal{T}\}$, $W = \{\mathbf{w}_t : t \in \mathcal{T}\}$, $X = \{\mathbf{x}_n\}_{n=1}^{N}$, and $C = \{\mathbf{c}_n\}_{n=1}^{N}$. The goal of the EM routine is to find a point estimate of $\theta = \{X, W, V\}$ that maximizes $\ln p(\mathcal{D}, \theta)$. We choose to marginalize out the path assignment distribution $p(C)$, which is a discrete distribution with infinite support, so as to make the problem more smooth.

According to our model,

$$\begin{aligned} p(\mathcal{D}, C, \theta) =& p(V)p(W)\left[\prod_{n=1}^{N} p(\mathbf{c}_n \mid V)p(\mathbf{x}_n \mid \mathbf{w}_{\mathbf{c}_n})\right] \\ &\times \prod_{r_{uv}\in\mathcal{D}} p(r_{uv} \mid \mathbf{x}_u, \mathbf{x}_v). \end{aligned}$$

We want to maximize the following objective:

$$\ln p(\mathcal{D}, \theta) = \underbrace{\sum_{C} q(C) \ln \frac{p(\mathcal{D}, C, \theta)}{q(C)}}_{L(\theta, q)} + \underbrace{\sum_{C} q(C) \ln \frac{q(C)}{p(C \mid \mathcal{D}, \theta)}}_{D_{\mathrm{KL}}(q \| p_\theta)}$$

The second term above is the Kullback-Leibler (KL) divergence from $p(C \mid \mathcal{D}, \theta)$ to an auxiliary distribution $q(C)$. The E-step of the EM algorithm aims to find a setting of $q(C)$ that makes the divergence zero. The first term $L(\theta, q)$ can be seen as an lower bound, which the M-step aims to maximize.

Each iteration of the EM algorithm proceeds by first running the E-step and then the M-step. Let $q_t(C)$ and θ_t be the result of the tth E-step and M-step, respectively. We then have

$$\begin{aligned} \ln p(\mathcal{D}, \theta_{t-1}) &= L(\theta_{t-1}, q) + D_{\mathrm{KL}}\left(q \| p_{\theta_{t-1}}\right) \\ &= L(\theta_{t-1}, q_t) + 0 && \text{(the } t\text{th E-step)} \\ &\le L(\theta_t, q_t) && \text{(the } t\text{th M-step)} \\ &\le L(\theta_t, q_t) + D_{\mathrm{KL}}\left(q_t \| p_{\theta_t}\right) \\ &= \ln p(\mathcal{D}, \theta_t) \end{aligned}$$

Therefore the EM algorithm will correctly converge to at least a local maxima of the objective.

3.2.1 E-Step. The goal of the E-step is to find an auxiliary distribution $q(C)$ such that $D_{\mathrm{KL}}(q \| p_\theta) = 0$. We can achieve this by setting $q(C) = p(C \mid \mathcal{D}, \theta)$. Therefore,

$$q(\mathbf{c}_n = \mathbf{c}) \propto S_{n,\mathbf{c}} \triangleq p(\mathbf{c}_n = \mathbf{c} \mid V)p(\mathbf{x}_n \mid \mathbf{w}_\mathbf{c}).$$

The tricky part is to compute the normalizing constant $\sum_{\mathbf{c}} S_{n,\mathbf{c}}$, because the path $\mathbf{c}$ can not only be a path inside the truncated tree, but also any path outside the truncated tree.

It is straightforward to compute $S_{n,\mathbf{c}}$ for an path completely inside the truncated tree. The problem is with the infinite number of paths outside the truncated tree. Let child(t) be the set of paths that stem from node t (a non-leaf node of the truncated tree) and are outside the truncated tree starting from t. It is then not hard to see that $\sum_{\mathbf{c}\in\text{child}(t)} S_{n,\mathbf{c}}$ can be efficiently computed using dynamic programming on the tree.

Once we have computed $S_{n,\mathbf{c}}$ for all the paths inside the truncated tree and $\sum_{\mathbf{c}\in\text{child}(t)} S_{n,\mathbf{c}}$ for all the non-leaf nodes of the truncated tree, we can obtain the normalizing constant $\sum_{\mathbf{c}} S_{n,\mathbf{c}}$ by summing these results. With the normalizing constant, other quantities such as $q(\mathbf{c}_n = \mathbf{c})$ and $\sum_{\mathbf{c}\in\text{child}(t)} q(\mathbf{c}_n = \mathbf{c})$ can be easily computed. These results are all we need for the following M-step.

For each vertex n, computing $\sum_{\mathbf{c}\in\text{child}(t)} S_{n,\mathbf{c}}$ for all $t \in \mathcal{T}$ costs $O(|\mathcal{T}|)$ when dynamic programming is used, where $|\mathcal{T}|$ is the number of the nodes in the truncated tree $\mathcal{T}$. Therefore an efficient implementation of the E-step has time complexity $O(N|\mathcal{T}|)$.

3.2.2 M-Step. The goal of the M-step is to maximize the lower bound $L(\theta, q)$ w.r.t. θ, under the auxiliary distribution $q(C)$ produced by the E-step. We use coordinate ascent for this purpose. We divide the optimization problem into three sub-problems.

Maximizing w.r.t. V. Maximizing the lower bound w.r.t. V is equivalent to maximizing the following sub-problem:

$$L(V) = \ln p(V) + \sum_{\mathbf{c}} \sum_{n=1}^{N} q(\mathbf{c}_n = \mathbf{c}) \ln p(\mathbf{c} \mid V)$$

The analytical solution can be found by setting the gradient to zero. Quantities such as $q(\mathbf{c}_n = \mathbf{c})$ and $\sum_{\mathbf{c}\in\text{child}(t)} q(\mathbf{c}_n = \mathbf{c})$ from the E-step are used to solve the summation over $\mathbf{c}$, so that the time compelexity for the summation is reduced to $O(|\mathcal{T}|)$. As a result, the time complexity for this sub-problem is $O(N|\mathcal{T}|)$ in total.

Maximizing w.r.t. W. Maximizing the lower bound w.r.t. W is equivalent to maximizing the following sub-problem:

$$L(W) = \ln p(W) + \sum_{\mathbf{c}} \sum_{n=1}^{N} q(\mathbf{c}_n = \mathbf{c}) \ln p(\mathbf{x}_n \mid \mathbf{w}_\mathbf{c}).$$

Setting $\sigma_w \to \infty$ is then equivalent to ignoring the first term $\ln p(W)$ above. This sub-problem can again be analytically solved in $O(N|\mathcal{T}|)$, in a way similar to the previous one.

Maximizing w.r.t. X. Maximizing the lower bound w.r.t. X is equivalent to maximizing the following sub-problem:

$$\begin{aligned} L(X) = &\sum_{n=1}^{N} \sum_{\mathbf{c}} q(\mathbf{c}_n = \mathbf{c}) \ln p(\mathbf{x}_n \mid \mathbf{w}_\mathbf{c}) \\ &+ \sum_{v:\, r_{uv}\in\mathcal{D}} \ln p(r_{uv} \mid \mathbf{x}_u, \mathbf{x}_v). \end{aligned}$$

We use conjugate ascent to optimize w.r.t. $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ sequentially. For each vertex, we run the conjugate ascent method for merely one step, as we found that it is sufficient to produce good results. The gradients can be computed in $O\left(N|\mathcal{T}| + |\mathcal{D}|\right)$ in total. The term $O\left(N|\mathcal{T}|\right)$ is due to the summation over n and $\mathbf{c}$, while the term $O\left(|\mathcal{D}|\right)$ is due to the summation over r_{uv}.

3.3 Refining the Truncated Tree

We adjust the structure of the truncated tree after every iteration of the EM algorithm. We adopt the strategy suggested by previous work [35]. It involves three operations (*grow*, *prune*, and *merge*). *Prune* and *merge* may theoretically lead to a decreased lower bound. However, we found that it rarely happens and the effect is negligible even when it does happen.

Grow. We compute $g(t) = \sum_{\mathbf{c} \in \text{child}(t)} \sum_{n=1}^{N} q(\mathbf{c}_n = \mathbf{c})$ for every non-leaf node t in the truncated tree. We then sample a new path stemming from a non-leaf node, according to $g(t)$.

Prune. We compute $\sum_{n=1}^{N} q(\mathbf{c}_n = \mathbf{c})$ for every path, and delete path $\mathbf{c}$ if $\frac{1}{N} \sum_{n=1}^{N} q(\mathbf{c}_n = \mathbf{c}) < \delta$. We set $\delta = 0.01$ in our experiments.

Merge. We measure the correlation between path i and path j with $\frac{\mathbf{p}_i^\top \mathbf{p}_j}{\|\mathbf{p}_i\| \|\mathbf{p}_i\|}$, where $\mathbf{p}_i = [q(\mathbf{c}_1 = i), q(\mathbf{c}_2 = i), \ldots, q(\mathbf{c}_N = i)]$. We remove one of the paths if the correlation is greater than 0.95.

4 EXPERIMENTS

We first assess the quality of the vertex representations on classification and link prediction. We then visualize the learned hierarchical taxonomy and the multiple levels of granularity. Finally, we investigate parameter sensitivity and scalability.

4.1 Experimental Setup

Baselines. We compare the performance of our approach with the following network embedding algorithms:

- DeepWalk [23]: DeepWalk generates a context window for each vertex from random walks and adopts SkipGram [20] to model the probability of a vertex appearing in the context. It then learns vertex representations by optimizing the SkipGram objective function.
- LINE [29]: It learns a $\frac{d}{2}$-dimensional representation to preserve first-order proximity (i.e. linked vertices tend to be similar) and another $\frac{d}{2}$-dimensional representation to preserve second-order proximity (i.e. vertices sharing common neighbors tend to be similar) for each vertex. It produces the final d-dimensional representation for a vertex by concatenating the two parts.
- node2vec [13]: It is a generalization of DeepWalk and uses a biased random walk sampler. The biased sampler can behave like either depth-first search (DFS) or breadth-first search (BFS), depending on its hyper-parameters.
- GraRep [7]: It first computes $\mathbf{P}, \mathbf{P}^2, \ldots, \mathbf{P}^K$, where $\mathbf{P}$ is the random walk transition matrix. It learns the kth ($1 \le k \le K$) $\frac{d}{K}$-dimensional vertex representations by factorizing $\mathbf{P}^k$, and concatenates the K parts to obtain the final d-dimensional representations.
- Walklets [24]: It is a multi-scale generalization of DeepWalk. It downsamples the random walks by keeping only every kth ($1 \le k \le K$) step to learn the kth $\frac{d}{K}$-dimensional vertex representations, and concatenates them to form the final d-dimensional representations.

Datasets. We evaluate the algorithms on a social network (BlogCatalog [30]), a protein-protein interaction network (PPI [5, 13]), and two citation networks (Cora and Citeseer [18]). The statistics of these networks are listed in Table 1, where the labels are user interests (BlogCatalog), biological states (PPI), and research areas (Cora and Citeseer), respectively.

Table 1: Statistics of the datasets.

Network	#Vertices	#Edges	#Labels
BlogCatalog	10,312	333,983	39
PPI	3,890	76,584	50
Cora	2,708	5,429	7
Citeseer	3,312	4,732	6

Hyper-parameters. We uniformly set the representation size d to 128. Some baselines, i.e. DeepWalk, node2vec and Walklets, generate positive observations from random walks, and augment the training set with negative observations using negative sampling [33]. For these methods, as in node2vec [13], we simulate 80 random walks of length 80 from each source vertex and sample five negative observations for each positive observations. For our algorithm, we set the hyper-parameter γ of the nCRP prior to one, which is the typical value used by most work that uses an nCRP. All of the other hyper-parameters of our approach and the baselines are tuned using grid search for each dataset separately. In particular, we experiment with $K \in \{1, 2, \ldots, 8\}$ for GraRep and Walklets, and choose the best setting for them. We found that our approach performs well on all of the datasets with $L = 4$, $\sigma_x = 0.50$, and $l = 2.00$. We hence report results of our approach under this same setting.

We repeat the experiments for ten times and report the averaged performance.

4.2 Classification

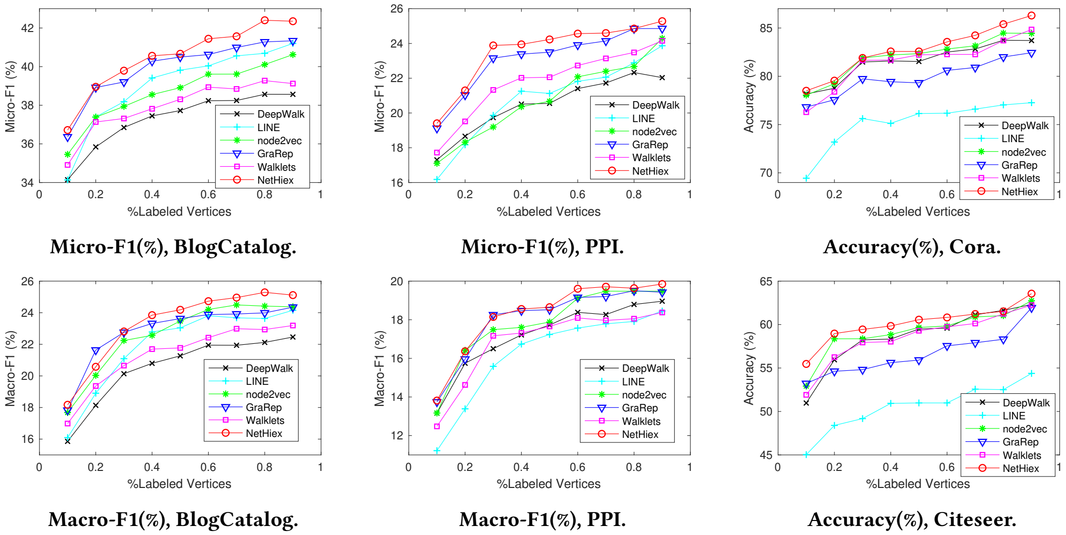
In this section, we evaluate our proposed method on the following two classification tasks:

- Multiclass classification: A vertex in Cora and Citeseer has exactly one label. We follow previous work [18, 34] and report the accuracy scores for this task.
- Multilabel classification: A vertex in BlogCatalog and PPI can have any number of labels. We follow previous work [13, 23, 30] and report the Macro-F1 and Micro-F1 scores.

Specifically, we train an one-vs-all logistic regression classifier on a subset of the vertices (treating their representations as features) and evaluate the performance of the classifier on half of the remaining vertices (with the other half serves as the validation set).

The results are shown in Figure 3. As we can see, our algorithm consistently outperforms all the baselines. We observe that preserving the multiple levels of granularity brings particularly significant improvement on BlogCatalog and PPI. GraRep and our approach are among the top performing approaches on these two datasets. And Walklets also performs better than its vanilla counterpart (i.e. DeepWalk).

On the other hand, the advantage of GraRep and Walklets over the other baselines is less significant on Cora and Citeseer. This is possibly because the labels (which are research areas) of Cora and Citeseer are roughly at the same scale, and predicting them does not necessitate multi-scale representations. Nevertheless, our



Micro-F1(%), BlogCatalog.

Micro-F1(%), PPI.

Accuracy(%), Cora.

Macro-F1(%), BlogCatalog.

Macro-F1(%), PPI.

Accuracy(%), Citeseer.

Figure 3: Classification results.

Table 2: The four strategies proposed by node2vec [13] for constructing edge representations.

Strategy	Symbol	Definition
Hadamard	$\circ$	$(\mathbf{x}_u \circ \mathbf{x}_v)_i = x_{u,i} x_{v,i}$
Average	$\boxplus$	$(\mathbf{x}_u \boxplus \mathbf{x}_v)_i = \frac{x_{u,i}+x_{v,i}}{2}$
Weighted-L1	$\|\cdot\|_{\bar{1}}$	$\|\mathbf{x}_u \cdot \mathbf{x}_v\|_{\bar{1},i} = \lvert x_{u,i} - x_{v,i} \rvert$
Weighted-L2	$\|\cdot\|_{\bar{2}}$	$\|\mathbf{x}_u \cdot \mathbf{x}_v\|_{\bar{2},i} = (x_{u,i} - x_{v,i})^2$

approach still outperforms all the baselines. This is probably because our approach jointly performs representation learning and hierarchical clustering, and thus the learned vertex representations are inherently more discriminative.

4.3 Link Prediction

There are two popular ways to perform link prediction with the learned vertex representations.

- Dot product: Many of the existing approaches, e.g. SDNE [36] and HOPE [21], measure the similarity between two vertices by computing the dot product of their representations. The higher the dot product is, the more similar they are. A high similarity score between two vertices indicates high probability of a link being formed between them.
- Edge features: node2vec [13] alternatively suggests fusing two d-dimensional vertex representations into a single d-dimensional edge representation, and training a binary classifier (whose input is the edge representation) to perform link prediction. The four strategies proposed by node2vec for constructing the edge representation is listed in Table 2.

We randomly remove a portion of links from the network for each dataset while ensuring every vertex has at least one neighbor, and run network embedding algorithms on the remaining part of the network to obtain vertex representations. The removed links are positive examples to be predicted. We then generate the negative examples by sampling an equal number of links that are not in the original network. The positive examples and the sampled negative examples form the test set.

We perform link prediction based on both of the dot-product and the edge-feature approach. We use a logistic regression classifier for the latter one, and train it with links in the remaining network. An equal number of negative links are again sampled to make the training set balanced. We follow previous work [13] and report the area under curve (AUC) scores.

First, we use Cora as an example to closely examine the results of the five different strategies (the dot-product approach and the four strategies used by node2vec) (the results on the other datasets follow a similar trend, and are omitted due to space constraints). To comprehensively evaluate the results, we also add another four traditional link prediction methods as baselines. The detailed results are shown in Table 3. As we can see, the Hadamard strategy generally performs best among the four strategies proposed by node2vec. However, the Hadamard strategy is not necessarily better than simply using the dot-product approach, even though theoretically the latter is a special case of the former. We conjecture that it is caused by the fact that training on the remaining edges for the edge classifier inevitably introduces bias. Therefore, the dot-product approach, which does not suffer from training bias, shows much more robust performance. Another interesting observation is that our algorithm performs best with the dot-product approach, even though our model is based on the Euclidean distance, rather than cosine similarity. It is possibly due to the fact that the two are closely related, i.e. $\|\mathbf{x}_u - \mathbf{x}_v\|^2 = \|\mathbf{x}_u\|^2 + \|\mathbf{x}_v\|^2 - 2\mathbf{x}_u^\top \mathbf{x}_v$. Our algorithm satisfies $\|\mathbf{x}_u\| \approx \|\mathbf{x}_v\|$ empirically. Therefore it is reasonable to use the dot-product approach for our algorithm.

Further, we report the best scores of the five different strategies on all the tested networks. The results are shown in Table 4. We can see that our approach consistently achieves significantly superior performance over all the baselines. Furthermore, the improvement

Table 3: The AUC scores on Cora, with different link prediction strategies. $\langle\cdot,\cdot\rangle$ represents the dot-product approach.

		%Missing Links				
Strategy	Algorithm	50%	40%	30%	20%	10%
$\circ$	DeepWalk	72.10	77.48	79.81	81.81	81.33
	LINE	66.95	70.96	74.00	75.90	77.94
	node2vec	72.63	77.17	79.34	81.39	82.42
	GraRep	72.76	76.21	80.39	83.57	82.31
	Walklets	71.02	75.11	78.51	81.03	81.65
	NetHiex	73.99	79.96	81.08	84.19	84.38
$\boxplus$	DeepWalk	58.87	58.46	58.53	59.11	59.71
	LINE	62.65	63.09	61.57	62.43	62.00
	node2vec	58.34	59.79	59.58	58.70	58.29
	GraRep	61.82	61.90	60.38	60.51	61.35
	Walklets	60.41	61.31	60.56	60.88	59.93
	NetHiex	53.67	54.57	53.55	54.92	54.03
$\|\cdot\|_{\bar{1}}$	DeepWalk	66.71	72.00	75.08	78.39	79.47
	LINE	65.19	68.73	71.80	74.09	76.20
	node2vec	67.39	72.06	74.47	78.03	79.58
	GraRep	69.00	74.35	76.96	80.05	79.69
	Walklets	68.33	72.31	74.76	78.34	78.60
	NetHiex	71.62	77.90	78.98	82.17	84.38
$\|\cdot\|_{\bar{2}}$	DeepWalk	67.63	72.68	75.95	79.11	80.02
	LINE	66.53	70.34	73.17	74.40	77.29
	node2vec	68.20	72.99	75.41	78.39	79.36
	GraRep	70.18	75.16	77.03	81.86	80.78
	Walklets	69.00	73.16	75.27	78.65	78.71
	NetHiex	73.64	79.51	81.62	85.18	85.48
$\langle\cdot,\cdot\rangle$	DeepWalk	74.50	80.48	81.59	84.28	84.22
	LINE	73.84	78.81	81.09	82.11	83.75
	node2vec	75.16	80.61	82.37	83.72	85.03
	GraRep	75.85	82.93	85.94	89.42	90.29
	Walklets	71.05	76.75	78.86	80.42	80.84
	NetHiex	**80.86**	**87.62**	**88.21**	**90.59**	**90.55**
(Results of Traditional Link Prediction Methods)						
Common Neighbors		62.37	66.98	70.79	72.13	73.84
Jaccard's Coefficient		62.35	66.88	70.69	72.07	73.82
Adamic-Adar Score		62.32	66.86	70.66	72.01	73.81
Pref. Attachment		70.60	71.32	70.77	70.31	69.82

over the best baseline is particularly remarkable when the number of the removed links is high, suggesting that our approach is robust to extremely sparse networks.

4.4 Hierarchical Taxonomy

We visualize the hierarchical taxonomy (see Figure 4) learned by our algorithm (with $L = 3$) on a word co-occurrence network, so as to intuitively understand the effects of the hierarchical taxonomy.

The word co-occurrence network is constructed from the titles of all the papers from the conferences listed in Table 5 published before the end of 2016. We collected the papers from dblp.org. We remove stop words by keeping the words whose TF-IDF scores are among the top 5% (the TF-IDF scores are computed by treating the conferences as documents and the paper titles as sentences). This leaves us a network with 1,262 vertices and 16,900 edges. For a category t at the lth layer, we list the five words that are closest to the category (in terms of $\|\mathbf{x}_{u,(l-1)\Delta d+1:l\Delta d} - \mathbf{w}_t\|$).

As we can see from Figure 4, our algorithm correctly identifies that there are seven research areas, even though we never explicitly specify the number of leaves. Our algorithm also considers bioinformatics, computational linguistics, and computer vision to be similar, in that machine learning techniques are widely adopted by them. Our algorithm therefore groups the three research areas under the same coarse-grained category, which is occupied by machine learning terminology, e.g. semi-supervised. Overall, the hierarchy shown in Figure 4 matches our intuition, and the finer-grained categories indeed contains words that are more specific than those of the coarser-grained categories.

4.5 Multiple Levels of Granularity

To better understand the effcacy of our algorithm, we follow Walklets' [24] approach and visualize the different levels of granularity in detail, as shown in Figure 5. A vertex s in Cora is randomly selected as the seed vertex. We then investigate the lth part ($1 \leq l \leq L+1$) of the learned vertex representations, i.e. $\mathbf{x}_{u,[(l-1)\Delta d+1]:\min(l\Delta d,d)}, u = 1, 2, \ldots, N$, to check whether it reflects a different level of granularity. More specifically, we compute the similarity between the seed vertex and the other vertices regarding the lth parts of their representations with

$$\mathrm{sim}_l(s,u) \triangleq \begin{cases} e^{-\|\mathbf{x}_{s,(l-1)\Delta d+1:l\Delta d} - \mathbf{x}_{u,(l-1)\Delta d+1:l\Delta d}\|^2}, & 1 \leq l \leq L, \\ e^{-\frac{\Delta d}{d-L\Delta d}\|\mathbf{x}_{s,L\Delta d+1:d} - \mathbf{x}_{u,L\Delta d+1:d}\|^2}, & l = L+1. \end{cases}$$

We then plot the heat maps of the similarity scores, as well as the distributions of the similarity scores, for $l = 1, 3, 5$ (note that we are using $L = 4$) in Figure 5.

Figure 5 suggests that our algorithm learns representations that are composed of a series of successively finer-grained components. Vertices that are considered similar can be many steps away from each other when looking at the coarser-grained components. On the other hand, vertices that are considered similar w.r.t. the finer-grained components tend to cluster in a small neighborhood.

4.6 Parameter Sensitivity

We examine the effect of the hyper-parameters of our model, taking the classification task on the Citeseer network as the example. As we can see from Figure 6, the performance generally improves when the dimension d of the representations increases. The improvement is more obvious when the number of labeled vertices is high, possibly because a high-dimensional space better preserves the diversity exhibited in a large sample size, while a small sample size benefits less from a high-dimensional space due to the risk of over-fitting.

The hyper-parameter σ_x controls how far away the vertices can be from their associated categories. We can see from Figure 6 that an ideal choice of σ_x should be small enough for the categories to play an important role. It should, however, not be too small, or it would limit the diversity within a category. The hyper-parameter l controls how far away two drastically different vertices should be from each other in the vector space. Similar to the pattern we observe for σ_x, a good choice of l should be moderately large. However, the

Table 4: Link prediction results.

			Baselines					This Work
Metric	Network	%Missing Links	DeepWalk	LINE	node2vec	GraRep	Walklets	NetHiex
AUC(%)	Citeseer	50%	77.00	77.25	77.58	74.11	74.57	**77.78**
		40%	79.76	80.36	80.04	76.02	78.09	**80.44**
		30%	82.12	82.41	83.03	81.55	80.80	**83.86**
		20%	82.97	84.00	83.02	85.81	83.86	**87.19**
		10%	86.59	88.44	86.74	87.15	86.16	**88.97**
	PPI	50%	74.60	73.23	75.13	76.81	74.55	**76.85**
		40%	75.00	74.34	75.92	77.73	74.19	**78.07**
		30%	75.49	75.13	76.02	77.80	76.37	**77.98**
		20%	76.73	75.35	77.04	78.51	77.89	**78.55**
		10%	77.30	75.69	77.69	78.96	78.89	78.96
	Cora	50%	74.50	73.84	75.16	75.85	71.05	**80.86**
		40%	80.48	78.81	80.61	82.93	76.75	**87.62**
		30%	81.59	81.09	82.37	85.94	78.86	**88.21**
		20%	84.28	82.11	83.72	89.42	81.03	**90.59**
		10%	84.22	83.75	85.03	90.29	81.65	**90.55**

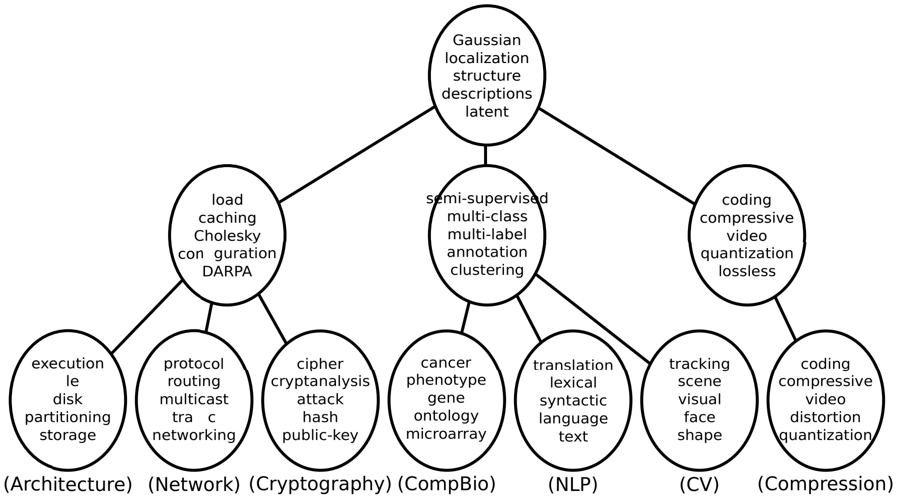

Gaussian localization structure descriptions latent
load caching Cholesky configuration DARPA
semi-supervised multi-class multi-label annotation clustering
coding compressive video quantization lossless
execution file disk partitioning storage
protocol routing multicast traffic networking
cipher cryptanalysis attack hash public-key
cancer phenotype gene ontology microarray
translation lexical syntactic language text
tracking scene visual face shape
coding compressive video distortion quantization
(Architecture)
(Network)
(Cryptography)
(CompBio)
(NLP)
(CV)
(Compression)

Figure 4: The hierarchical taxonomy learned by NetHiex, on a word co-occurrence network constructed from the titles of computer science papers (with stop words removed). We list the top five words that have the most similar representations to the category representation for each category. Our algorithm correctly identifies that there are seven research areas.

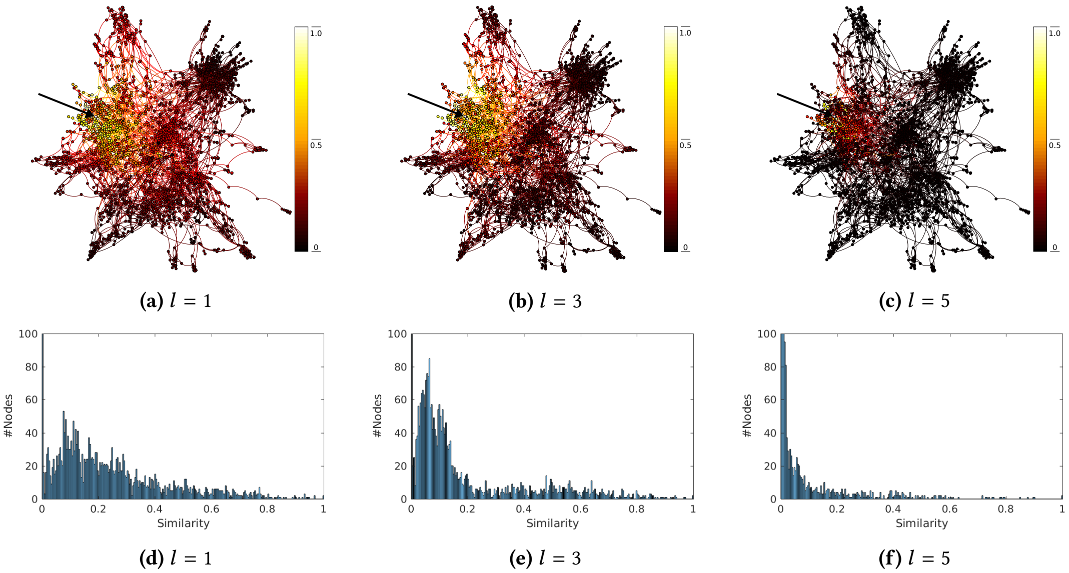
Table 5: The conferences from which we collect the publications and construct the word co-occurrence network. The taxonomy learned for the network is visualized in Figure 4.

Research Area	Conferences
Computer Architecture	SC, ISCA
Computer Network	INFOCOM, SIGCOMM
Cryptography	EUROCRYPT, CRYPTO
Bioinformatics	BIBM, RECOMB
Computational Linguistics	ACL, COLING
Computer Vision	CVPR, ICCV
Data Compression	DCC

effect of l is less significant when the number of labeled vertices is small. This is possibly because capturing the differences between individual vertices (with a proper choice of l) is much less helpful than finding the underlying hierarchical taxonomy (with a proper σ_x) when the labeled data is scarce.

4.7 Scalability

We found empirically that our EM algorithm is close to convergence after roughly ten iterations, and all our experiments finished with less than twenty iterations. Each iteration of our algorithm is linear in the network size, making our algorithm as scalable as the existing algorithms such as DeepWalk. To further support the claim, we



(a) $l = 1$ **(b)** $l = 3$ **(c)** $l = 5$

(d) $l = 1$ **(e)** $l = 3$ **(f)** $l = 5$

Figure 5: The distances of the vertices from a randomly selected seed vertex (pointed to by the arrows in (a)–(c)) in Cora, in terms of the 1st, 3rd, and 5th parts of their representations, respectively. (a)–(c) visualize the distance as heat maps. (d)–(f) are the distributions of the distances. The different parts of the representations clearly exhibit different levels of granularity.

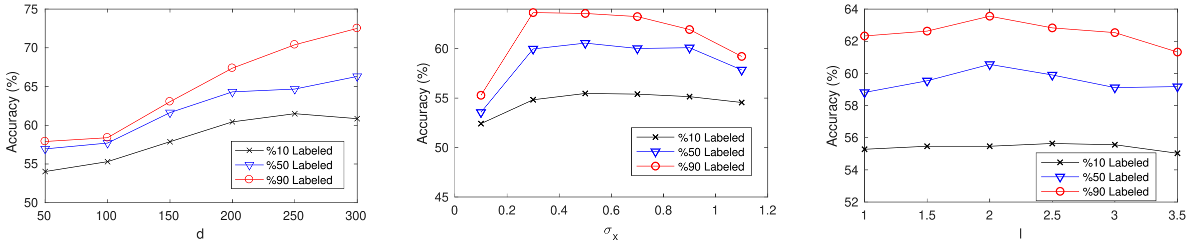


Figure 6: Parameter sensitivity on Citeseer.

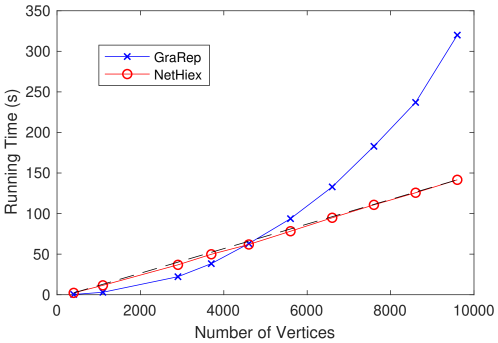


Figure 7: NetHiex scales linearly with the network size.

generate a series of random networks according to the Barabási-Albert model, with increasing sizes from 400 to 10,000 vertices and a constant average degree of 30. As shown in Figure 7, our algorithm scales linearly with the network size.

5 RELATED WORK

Network embedding [6, 11, 14], also known as network representation learning, gains momentum after the recent success of DeepWalk [23]. Unlike the eigendecomposition-based predecessors (e.g. LLE [26], Laplacian eigenmaps [2], and Isomap [32]), DeepWalk combines word2vec [19] and random walking, and permits scalable stochastic optimization. Many new algorithms are presented to better capture the network structure since then. LINE [29] proposes to preserve first-order and second-order proximity, and node2vec [13] aims to explore diverse neighborhoods with biased random walks. Other concepts, such as nonlinearity [8, 36], higher-order relationships [7, 21, 24], network communities [38], and structural identity [25], are also explored. Another line of research focuses on embedding more complex networks, e.g. attributed networks [40], directed networks [21], signed networks [37, 43], heterogeneous information networks [9, 10, 12], and dynamic networks [16, 17, 42, 44]. Network embedding has also been integrated into other paradigms, e.g. semi-supervised learning [41] and inductive learning [15]. We are, however, not aware of other work on network representation learning that studies the underlying hierarchical taxonomy.

The hierarchies of entities are more actively studied in the field of natural language processing (NLP). For example, ontology learning aims to automatically extract ontologies from text [39], where the hierarchy of the extracted concepts is constructed. The closest work to ours from NLP is probably hierarchical topic modeling [4]. In hierarchical topic models, the topics form a hierarchy and each document is associated with one [4] or more [22] paths (by nesting CRPs [1] and hierarchical Dirichlet processes [31], respectively). Our model is also built on the nCRP [4]. However, the way we leverage the nCRP is different. Furthermore, previous nCRP-based models typically adopt MCMC [3, 4] or variational inference (VI) [35] for optimization, while we present an EM algorithm, which is much faster than MCMC and easier to implement than VI.

6 CONCLUSION

In this paper, we have presented NetHiex, a network embedding algorithm that reveals and leverages the hierarchical taxonomy. In particular, we leverage the hierarchical taxonomy to capture the different levels of granularity and alleviate data scarcity.

Extending our work to weighted/attributed/directed networks is straightforward (see section 2.3). It is, however, less clear how to generalize our algorithm to heterogeneous information networks (HINs). An interesting direction for future work is to integrate meta paths [10, 12, 28] and the underlying hierarchical taxonomy, to learn vertex representations for a HIN.

ACKNOWLEDGMENTS

This work is supported by the National Program on Key Basic Research Project (No. 2015CB352300), and the National Natural Science Foundation of China (No. 61702296, No. 61772304, No. 61521002, No. 61531006, and No. U1611461). Thanks for the research fund of the Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and the Young Elite Scientist Sponsorship Program by CAST.

REFERENCES

[1] D. Aldous. 1985. Exchangeability and Related Topics. *Ecole d'Ete de Probabilities de Saint-Flour XIII 1983* (1985).

[2] Mikhail Belkin and Partha Niyogi. 2001. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *Proceedings of NIPS 2001.*

[3] D. Blei, T. Griffiths, and M. Jordan. 2010. The Nested Chinese Restaurant Process and Bayesian Nonparametric Inference of Topic Hierarchies. *J. ACM* (2010).

[4] D. Blei, T. Griffiths, M. Jordan, and J. Tenenbaum. 2003. Hierarchical Topic Models and the Nested Chinese Restaurant process. In *Proceedings of NIPS 2003.*

[5] Bobby-Joe Breitkreutz, Chris Stark, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, Michael Livstone, Rose Oughtred, Daniel H. Lackner, JÃijrg BÃďhler, Valerie Wood, Kara Dolinski, and Mike Tyers. 2008. The BioGRID Interaction Database: 2008 update. *Nucleic Acids Research* (2008).

[6] Hongyun Cai, Vincent W. Zheng, and Kevin C.-C. Chang. 2018. A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. *TKDE'18* (2018).

[7] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning Graph Representations with Global Structural Information. In *Proceedings of CIKM 2015.*

[8] Shaosheng Cao, Wei Lu, and Xiongkai Xu. 2016. Deep Neural Networks for Learning Graph Representations. In *Proceedings of AAAI 2016.*

[9] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C. Aggarwal, and Thomas S. Huang. 2015. Heterogeneous Network Embedding via Deep Architectures. In *Proceedings of KDD 2015.*

[10] Ting Chen and Yizhou Sun. 2017. Task-Guided and Path-Augmented Heterogeneous Network Embedding for Author Identification. In *Proceedings of WSDM 2017.*

[11] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2017. A Survey on Network Embedding. (2017). arXiv:cs.SI/1711.08752

[12] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. 2017. Metapath2Vec: Scalable Representation Learning for Heterogeneous Networks. In *Proceedings of KDD 2017.*

[13] Aditya Grover and Jure Leskovec. 2016. Node2Vec: Scalable Feature Learning for Networks. In *Proceedings of KDD 2016.*

[14] W. Hamilton, R. Ying, and J. Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin* (2017).

[15] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of NIPS 2017.*

[16] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed Network Embedding for Learning in a Dynamic Environment. In *Proceedings of CIKM 2017.*

[17] Jianxin Ma, Peng Cui, and Wenwu Zhu. 2018. DepthLGP: Learning Embeddings of Out-of-Sample Nodes in Dynamic Networks. In *Proceedings of AAAI 2018.*

[18] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the Construction of Internet Portals with Machine Learning. *Information Retrieval Journal* (2000).

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

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

[21] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric Transitivity Preserving Graph Embedding. In *Proceedings of KDD 2016.*

[22] John Paisley, Chong Wang, David M. Blei, and Michael I. Jordan. 2015. Nested Hierarchical Dirichlet Processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2015).

[23] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of KDD 2014.*

[24] Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. 2017. Don't Walk, Skip! Online Learning of Multi-scale Network Embedding. In *Proceedings of ASONAM 2017.*

[25] Leonardo F. R. Ribeiro, Pedro H. P. Saverese, and Daniel R. Figueiredo. 2017. struc2vec: Learning Node Representations from Structural Identity. In *Proceedings of KDD 2017.*

[26] Sam Roweis and Lawrence Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* (2000).

[27] J. Sethuraman. 1994. A Constructive Definition of Dirichlet Priors. *Statistica Sinica* (1994).

[28] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. In *Proceedings of VLDB 2011.*

[29] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *Proceedings of WWW 2015.*

[30] Lei Tang and Huan Liu. 2009. Relational Learning via Latent Social Dimensions. In *Proceedings of KDD 2009.*

[31] Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. 2006. Hierarchical Dirichlet processes. *J. Amer. Statist. Assoc.* (2006).

[32] J. B. Tenenbaum, V. de Silva, and J. C. Langford. 2000. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science* (2000).

[33] Kai Chen Greg Corrado Tomas Mikolov, Ilya Sutskeve and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Proceedings of NIPS 2013.*

[34] Cunchao Tu, Weicheng Zhang, Zhiyuan Liu, and Maosong Sun. 2016. Max-Margin DeepWalk: Discriminative Learning of Network Representation. In *Proceedings of IJCAI 2016.*

[35] Chong Wang and David M. Blei. 2009. Variational Inference for the Nested Chinese Restaurant Process. In *Proceedings of NIPS 2009.*

[36] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural Deep Network Embedding. In *Proceedings of KDD 2016.*

[37] Suhang Wang, Jiliang Tang, Charu Aggarwal, Yi Chang, and Huan Liu. 2017. Signed Network Embedding in Social Media. In *Proceedings of SDM 2017.*

[38] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding. In *Proceedings of AAAI 2017.*

[39] Wilson Wongx, Wei Liu, and Mohammed Bennamoun. 2012. Ontology Learning from Text: A Look back and into the Future. *Comput. Surveys* (2012).

[40] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y. Chang. 2015. Network Representation Learning with Rich Text Information. In *Proceedings of IJCAI 2015.*

[41] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-supervised Learning with Graph Embeddings. In *Proceedings of ICML 2016.*

[42] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. TIMERS: Error-Bounded SVD Restart on Dynamic Networks. In *Proceedings of AAAI 2018.*

[43] Q. Zheng and D.B. Skillicorn. 2015. Spectral Embedding of Signed Networks. In *Proceedings of SDM 2015.*

[44] Dingyuan Zhu, Peng Cui, Ziwei Zhang, Jian Pei, and Wenwu Zhu. 2018. High-order Proximity Preserved Embedding For Dynamic Networks. *TKDE'18* (2018).