Hierarchical Taxonomy Aware Network Embedding

Jianxin Ma
Tsinghua University
majx13fromthu@gmail.com

Xiao Wang
Tsinghua University
wangxia007@mail.tsinghua.edu.cn

Peng Cui
Tsinghua University
cuip@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 reflect 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:

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,
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 multiple 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 for real world large-scale networks. Suitable for real world large-scale networks.

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 $c_i$ be the path of vertex $n$. The path $c_i$ 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

$$p(c_1, c_2, \ldots, c_N).$$

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. $\{x_n\}_{n=1}^N \subset \mathbb{R}^d$, as well as the representations of the categories in the tree hierarchy, i.e. $\{w_t : \text{category } t \text{ in the hierarchy} \} \subset \mathbb{R}^{\Delta d}$, where $\Delta d = \left\lfloor \frac{d}{\mathcal{L}+1} \right\rfloor$.

The prior distribution of the category representations is

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

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

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

$$x_{n,1:L\Delta d} \sim \text{Normal} \left( w_{c_n}, \sigma_x^2 I \right).$$

---

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 $L$. We further examine the hierarchical taxonomy learned on a word co-occurrence network, and visualize the multiple levels of granularity.
where $\sigma_i$ is a hyper-parameter, and $w_{c_n}$ is the concatenation of all the $w_j$ visited by the path $c_n$. We then let the $(L + 1)$th part of $x_n$ follow Normal$(0, \infty)$, i.e. we do not regularize the last part.

Let $D = \{r_{uv}\}$ 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 there is certain that there is no link between $u$ and $v$. Our model assumes that

$$r_{uv} \sim \text{Bernoulli}(e^{-\frac{||u-v||^2}{l}}),$$

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, so as to preserve asymmetric transitivity [21]. For example, we can let $y_u \sim \text{Normal}(0, \sigma_u^2 I)$ and $r_{uv} \sim \text{Bernoulli}(\sigma(x_u^T y_v))$, 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 $c_n$ heavily depends on $c_{1:(n-1)}$, 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 $\gamma$, i.e. $v_i \sim \text{Beta}(1, \gamma)$. Each customer then independently chooses the $i$th 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 $w_t$ for all the nodes outside the truncated tree to the means of their prior distributions ($p(v_t)$ and $p(w_t)$), while allowing the $v_t$ and $w_t$ 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 $V = \{v_t : t \in \mathcal{T}\}$, $W = \{w_t : t \in \mathcal{T}\}$, $X = \{x_n\}_{n=1}^N$, and $C = \{c_n\}_{n=1}^N$. The goal of the EM routine is to find a point estimate of $\theta = \{X, W, V\}$ that maximizes $\ln p(D, \theta)$. 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,

$$p(D, C, \theta) = p(V) p(W) \prod_{n=1}^N p(c_n \mid V) p(x_n \mid w_{c_n}) \times \prod_{r_{uv} \in E} p(r_{uv} \mid x_u, x_v).$$

We want to maximize the following objective:

$$\ln p(D, \theta) = \sum_C q(C) \ln \frac{p(D, C, \theta)}{q(C)} + \sum_C q(C) \ln \frac{q(C)}{p(C \mid D, \theta)} + D_{KL}(q \mid p_{\theta}).$$

The second term above is the Kullback-Leibler (KL) divergence from $p(C \mid D, \theta)$ 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 $q(C)$ and $\theta_t$ be the result of the $t$th E-step and M-step, respectively. We then have

$$\ln p(D, \theta_{t-1}) = L(\theta_{t-1}, q) + D_{KL}(q \mid p_{\theta_{t-1}}) = L(\theta_t, q_t) + 0 = L(\theta_t, q_t) + D_{KL}(q_t \mid p_{\theta_t}) = \ln p(D, \theta_t)$$

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 $D_{KL}(q \mid p_{\theta}) = 0$. We can achieve this by setting $q(C) = p(C \mid D, \theta)$. Therefore,

$$q(c_n = c) \triangleq S_{n,c} \triangleq p(c_n = c \mid V)p(x_n \mid w_c).$$

The tricky part is to compute the normalizing constant $\sum_c S_{n,c}$, because the path $c$ can not only be a path inside the truncated tree, but also any path outside the truncated tree. It is straightforward to compute $S_{n,c}$ for an path completely inside 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 $\sum_{c \in child(t)} S_{n,c}$ can be efficiently computed using dynamic programming on the tree.

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

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

3.2.2 M-Step. The goal of the M-step is to maximize the lower bound $L(\theta, q)$ w.r.t. $\theta$, 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:

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

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

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

$$L(W) = \ln p(W) + \sum_{c \in C} \sum_{n=1}^N q(c_n = c) \ln p(x_n \mid w_c).$$

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 $O(N|T|)$, 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:

$$L(X) = \sum_{n=1}^N \sum_{c \in C} q(c_n = c) \ln p(x_n \mid w_c) + \sum_{u \in U \mid r_{uv} \in D} \ln p(r_{uv} \mid x_u, x_v).$$

We use conjugate ascent to optimize w.r.t. $x_1, x_2, \ldots, x_N$ 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 $O(|D|)$ in total. The term $O(N|T|)$ is due to the summation over $n$ and $c$, while the term $O(|D|)$ 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 \( g(t) = \sum_{c \in \text{child}(t)} \sum_{n=1}^{N} q(c_n = c) \) 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 \( \sum_{n=1}^{N} q(c_n = c) \) for every path, and delete path \( c \) if \( \frac{1}{N+1} \sum_{n=1}^{N} q(c_n = c) < \delta \). We set \( \delta = 0.01 \) in our experiments.

Merge. We measure the correlation between path \( i \) and path \( j \) with \( p_i/p_j \), where \( p_i = [q(e_1 = i), q(e_2 = i), \ldots, q(e_N = i)] \). 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 with the following networks are listed in Table 1, where the labels are user 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 \( k \)th \( (1 \leq k \leq K) \) \( \frac{d}{K} \)-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 \( k \)th \((1 \leq k \leq K)\) step to learn the \( k \)th \( \frac{d}{K} \)-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.

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 \( \gamma \) 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 \( K \in \{1, 2, 4, 8\} \) 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 \( I = 2.00 \). 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.

We specifically 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

---

Table 1: Statistics of the datasets.

<table>
<thead>
<tr>
<th>Network</th>
<th>#Vertices</th>
<th>#Edges</th>
<th>#Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlogCatalog</td>
<td>10,312</td>
<td>333,983</td>
<td>39</td>
</tr>
<tr>
<td>PPI</td>
<td>3,890</td>
<td>76,584</td>
<td>50</td>
</tr>
<tr>
<td>Cora</td>
<td>2,708</td>
<td>5,429</td>
<td>7</td>
</tr>
<tr>
<td>Citeseer</td>
<td>3,312</td>
<td>4,732</td>
<td>6</td>
</tr>
</tbody>
</table>
Table 2: The four strategies proposed by node2vec [13] for constructing edge representations.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadamard</td>
<td>⊙</td>
<td>$(x_u \odot x_v)<em>i = x</em>{ui} x_{vi}$</td>
</tr>
<tr>
<td>Average</td>
<td>⋂</td>
<td>$(x_u \bowtie x_v)<em>i = \frac{x</em>{ui} + x_{vi}}{2}$</td>
</tr>
<tr>
<td>Weighted-L1</td>
<td>⌄</td>
<td>$|x_u \cdot x_v|_1,i =</td>
</tr>
<tr>
<td>Weighted-L2</td>
<td>⌆</td>
<td>$|x_u \cdot x_v|<em>2,i = (x</em>{ui} - x_{vi})^2$</td>
</tr>
</tbody>
</table>

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 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. $\|x_u - x_v\|^2 = \|x_u\|^2 + \|x_v\|^2 - 2x_u^T x_v$. Our algorithm satisfies $\|x_u\| = \|x_v\|$ 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
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 \[ \| x_u, t : l−1 \|_{L + 1} - \| x_u, t \|_{L + 1} \]).

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 labels. 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 efficacy 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 \( l \)th part \( (1 ≤ l ≤ L + 1) \) of the learned vertex representations, i.e. \( x_u, t : 1 ≤ l ≤ L + 1 \), \( u = 1, 2, \ldots, N \), 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 \( l \)th parts of their representations with

\[
\text{sim}_l(s, u) \triangleq \begin{cases} 
\frac{1}{N} \sum_{d = 1}^{\min(L, d)} \| x_u, t : l−1 \|_{L + 1} - \| x_u, t \|_{L + 1}, \\
\frac{1}{\sqrt{N}} \sum_{d = 1}^{\min(L, d)} \| x_u, t : l−1 \|_{L + 1} - \| x_u, t \|_{L + 1}, 
\end{cases} \quad l = 1, \ldots, L.
\]

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 \( \sigma_x \) controls how far away the vertices can see from Figure 4, the performance generally improves when 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 \( \sigma_x \) controls how far away two drastically different vertices should be from each other in the vector space. Similar to the pattern we observe for \( \sigma_x \), a good choice of \( l \) should be moderately large. However, the
Table 4: Link prediction results.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Network</th>
<th>%Missing Links</th>
<th>Baselines</th>
<th>This Work</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>DeepWalk</td>
<td>LINE</td>
</tr>
<tr>
<td>AUC(%)</td>
<td>Citeseer</td>
<td>50%</td>
<td>77.00</td>
<td>77.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40%</td>
<td>79.76</td>
<td>80.36</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30%</td>
<td>82.12</td>
<td>82.41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20%</td>
<td>82.97</td>
<td>84.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10%</td>
<td>86.59</td>
<td>88.44</td>
</tr>
<tr>
<td></td>
<td>PPI</td>
<td>50%</td>
<td>74.60</td>
<td>73.23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40%</td>
<td>75.00</td>
<td>74.34</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30%</td>
<td>75.49</td>
<td>75.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20%</td>
<td>76.73</td>
<td>75.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10%</td>
<td>77.30</td>
<td>75.69</td>
</tr>
<tr>
<td></td>
<td>Cora</td>
<td>50%</td>
<td>74.50</td>
<td>73.84</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40%</td>
<td>80.48</td>
<td>78.81</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30%</td>
<td>81.59</td>
<td>81.09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20%</td>
<td>84.28</td>
<td>82.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10%</td>
<td>84.22</td>
<td>83.75</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Research Area</th>
<th>Conferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Architecture</td>
<td>SC, ISCA</td>
</tr>
<tr>
<td>Computer Network</td>
<td>INFOCOM, SIGCOMM</td>
</tr>
<tr>
<td>Cryptography</td>
<td>EUROCRYPT, CRYPTO</td>
</tr>
<tr>
<td>Bioinformatics</td>
<td>BIBM, RECOMB</td>
</tr>
<tr>
<td>Computational Linguistics</td>
<td>ACL, COLING</td>
</tr>
<tr>
<td>Computer Vision</td>
<td>CVPR, ICCV</td>
</tr>
<tr>
<td>Data Compression</td>
<td>DCC</td>
</tr>
</tbody>
</table>

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...
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