A Deep Learning Framework for Self-evolving Hierarchical Community Detection

Daizong Ding[1], Mi Zhang[1*], Hanrui Wang[1], Xudong Pan[1], Min Yang[1], Xiangnan He[2]
[1]School of Computer Science, Fudan University
{17110240010,mi_zhang,18212010031,18110240010,m_yang}@fudan.edu.cn
[2]School of Data Science, University of Science and Technology of China
xiangnanhe@gmail.com

ABSTRACT

Hierarchical community detection, which aims at discovering the hierarchical structure of a graph, attracts increasing attention due to its wide range of applications. However, due to the difficulty of parametrizing the community tree, existing methods mainly rely on heuristic algorithms, which are limited by their low accuracy and inability to handle new observations. As far as we know, how to leverage deep learning techniques to better discover hierarchical communities remains almost blank in the existing literature. In this paper, we present the first deep learning framework called ReinCom for hierarchical community detection. To address the challenge of parametrizing the community tree, we propose a novel growing-up process where, at each step, we first partition nodes into the community tree and then adjust the community tree according to the partition results. To learn an optimal growing-up process, we propose an embedding agent and a community agent to implement the two sub-steps respectively. Furthermore, we also propose an online learning strategy for new observations on the graph. Empirical results show that our proposed model has better modeling effectiveness than the state-of-the-art methods. For example, in terms of modularity, the performance of ReinCom is 33% higher than previous community detection works. Besides, with the aid of the learned node embeddings, we also devise a graph visualization algorithm which can consistently reflect the latent hierarchical structure of a graph.

CCS CONCEPTS

• **Information systems** → *Clustering*.

KEYWORDS

Hierarchical Community Detection, Deep Learning

ACM Reference Format:
Daizong Ding[1], Mi Zhang[1*], Hanrui Wang[1], Xudong Pan[1], Min Yang[1], Xiangnan He[2]. 2021. A Deep Learning Framework for Self-evolving Hierarchical Community Detection. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482223

[*]The corresponding author is Mi Zhang.

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482223

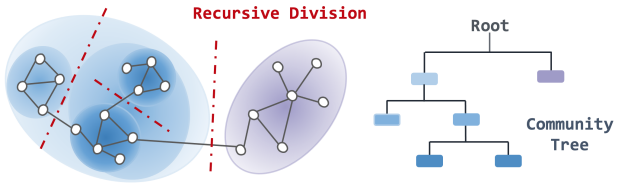

Figure 1: The hierarchical community detection task and existing heuristic algorithms.

1 INTRODUCTION

Community detection, which aims at partitioning nodes on a graph to several communities, can be widely used in various graph-based application, e.g., grouping researchers with similar interests in an author collaboration network [51] and clustering web-pages according to the hyperlinks [49]. Due to its wide range of applications, community detection has attracted increasing attention from both academia and industry. Existing community detection methods can be mainly classified into two categories [23, 50]: non-hierarchical community detection methods such as Spectral Clustering [27, 34] and Block Modeling [1, 39], and hierarchical community detection methods such as Label Propagation [51, 58] and Louvain algorithm [3]. Hierarchical methods aim to learn hierarchical relations between communities (i.e., *community tree*). Since complex networks in real-world applications usually have hierarchical structures [7, 10], e.g., certain content integration websites like Yahoo usually cover a wide range of topics and have hyperlinks to web pages on more specific topics, such hierarchical modeling is able to characterize this property and leads to better community detection results [25]. Therefore, in this paper, we focus on *hierarchical community detection*. Existing methods mainly use heuristic algorithms [9, 14, 21] to detect hierarchical structure. For example, [9] proposes to recursively divide a large community into smaller communities through a greedy strategy, while [3, 13] propose to recursively merge small communities into larger communities. After running the pre-defined heuristic algorithms, one can obtain a community tree and the corresponding node-community affiliation, as shown in Fig. 1.

Recently, with the rapid development of deep learning, deep neural networks (DNN) have been applied to various graph applications, such as Graph Convolutional Network (GCN) for node classification [18, 54] and Neural Graph Collaborative Filtering

(NGCF) for recommender systems [47, 48]. Compared with traditional methods, a major advantage of DNNs is that they can learn effective node representations through deep nonlinear structures, which enables them to extract complex topological patterns of a graph. However, as far as we know, deep learning has not been successfully applied to hierarchical community detection yet. This is mainly because we usually require parametrized inputs and outputs to design deep learning models. For example, in node classification problems, the input (i.e., node) and the output (i.e., label) of a model can be defined as an integer number and a one-hot vector respectively. However, this requirement can not be fulfilled for hierarchical community detection, whose output is by definition a community tree of uncertain width and depth [20, 37]. As a result, despite the effectiveness of deep learning techniques, *how to leverage them to better discover hierarchical communities remains almost blank in the existing literature.*

In this paper, we present our novel deep learning framework called **ReinCom** to bridge this gap. To tackle the challenge of parametrizing the community tree, we divide the generation of a community tree into two sub-processes: **(1)** learning the node-community affiliation given a community tree and **(2)** further adjusting the community tree according to the partition results. Based on this, we design a deep learning model to implement the two sub-processes above. To the best of our knowledge, ReinCom is the first solution which applies deep learning model to hierarchical community detection. In comparison to traditional methods, the major advantages of our methods lie in the following aspects:

- **Effectiveness of Hierarchical Modeling:** ReinCom is more effective in discovering the latent hierarchical structure of a graph with the aid of deep learning techniques. From our perspective, the reasons are two-folds. First, we leverage graph embeddings to model node-community affiliation, which can characterize local structure effectively by the low-dimensional space. Second, we introduce reinforcement learning to search the optimal community tree, which can balance the exploration and exploitation well compared with heuristic algorithms. Empirical results show that in most cases ReinCom outperforms the state-of-the-art hierarchical community detection methods by a noticeable margin. For example, ReinCom makes 33% relative improvements in terms of modularity on Wiki-Vote.
- **Online Updating for New Observations:** When there are new nodes and edges, existing methods need to be re-run on the whole graph [30], e.g., recursively dividing or merging, which often leads to a totally different community tree due to the instability of heuristic algorithms. To date, how to perform effective updating for new observations without perturbing the existing community tree is still an open problem. As real-world graphs often have new incoming observations, there is an urging need to devise a novel hierarchical community detection method to embrace this important characteristic [7, 10]. In this work, with the aid of the novel two sub-processes, we propose an online updating strategy for new observations. When new nodes and edges are added to the graph, ReinCom can make slight adjustments to the existing community tree and fine-tune the node-community affiliation in order to maintain the modeling effectiveness. We conduct extensive experiments to validate the effectiveness of our online updating strategy.
- **Application to Multiple Downstream Tasks:** Meanwhile, ReinCom also learns effective node embeddings, which can be used in conjunction with a wide range of downstream graph-based applications, including link prediction, node classification and graph visualization. We validate the quality of the learned embeddings in various graph-based tasks. Besides, we develop a graph visualization algorithm which is able to consistently reflect the latent hierarchical structure of a graph.

2 RELATED WORKS

Community detection aims at finding latent node-community affiliation relations and detecting the modular structure of a graph [12]. Different from standard clustering algorithms which group data according to their content information [52], e.g., images used in K-nearest neighbors (KNN) [8] and node properties used in Graph Neural Network (GNN) [18, 55], community detection algorithms need to find communities based on the discrete linkage information in a graph. Existing community detection methods can be roughly classified into two categories [23, 50]: (1) Non-hierarchical community detection methods such as Clique Percolation Method (CPM) [11, 29], random walk [16], spectral clustering [27, 42, 57] and blocking models [26, 53]; (2) Hierarchical community detection methods such as Label Propagation Algorithm (LPA) [51, 58] and Girvan-Newman Algorithm [43]. For a more comprehensive introduction, please refer to [12, 50]. Compared with non-hierarchical methods, hierarchical community detection methods aim at learning to build a community tree through partitioning nodes to communities on the tree [2, 19, 21, 25, 31]. This design helps them model the hierarchical topology of complex networks in real-world applications well [7, 10]. In this paper, we focus on hierarchical community detection (HCD).

Despite the advantages of being able to detect hierarchical structures, such modeling however increases the complexity of the optimization. For instance, finding an optimal tree structure is proved to be a typical NP-hard problem [37]. To address the issue, most methods propose to use heuristic algorithms like greedy strategy [21], e.g., to recursively divide a large community into smaller communities [9] or to recursively merge small communities into larger communities [3, 13]. Although some of them try to use Bayesian non-parametrics to infer the tree structure [4], they could only be applied to small networks due to the high time complexity [19].

Recently, deep learning is widely used in graph based tasks, e.g., Graph Convolutional Network (GCN) and Graph Attention Network (GAT) for node classification [18, 44], Graph Neural Network (GNN) for Knowledge Base Completion [15] and Neural Graph Collaborative Filtering (NGCF) for recommender systems [45, 47, 48]. This is mainly because its benefits of learning nonlinear features and data representations at multiple levels of abstraction, which is suitable for complex topological graphs. However, it is difficult to apply the technique to HCD because a neural network model usually needs an output form which is clearly parametrized, e.g., one-hot label in node classification or rating matrix in recommender systems. Yet, this requirement could hardly be met in HCD, mainly because its output is usually a community tree with unknown width

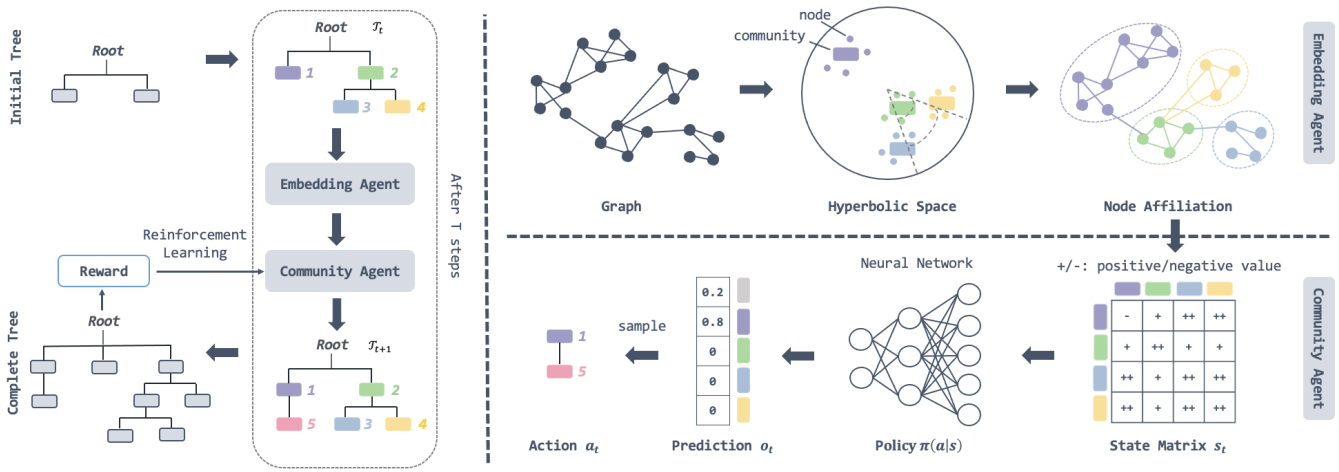

Figure 2: An Overview of our proposed framework ReinCom, the growing-up process of a community tree (Left) and our general design of the embedding agent and the community agent (Right).

and depth [20]. Therefore, how to leverage deep learning techniques in HCD remains an open problem.

3 GENERATING A COMMUNITY TREE

3.1 The Growing-up Process

We first formally define the task of hierarchical community detection. Formally, we denote an undirected homogeneous graph $\mathcal{G} = (\mathcal{V}, \Upsilon)$ with $|\mathcal{V}| = N$ nodes and the linkage information $y_{ij} = \{0, 1\} \in \Upsilon$, which describes whether a link is observed between $v_i, v_j \in \mathcal{V}$. We assume $\mathcal{T}_t$ is a tree of t communities. For a node $v_i \in \mathcal{V}$, we assign it with a community $c_i \in \{1, \cdots, t\}$. In other words, we call the node v_i is affiliated with the community c_i and c_i is the node-community affiliation. The goal of hierarchical community detection is to *find an optimal tree structure $\mathcal{T}^*$ and to learn node-community affiliation c_i* according to the linkage information Υ. Specifically, for those $y_{ij} = 1$, the learned communities c_i and c_j should be close on $\mathcal{T}^*$ and vice versa.

Previous methods rely on heuristic algorithms to find a solution, e.g., greedy strategy and genetic algorithms, making them unable to search better community tree structure and handle new observations. To address these issues, we take the lead to introduce deep learning techniques to this traditional task. Towards the challenge of parametrizing the community tree, we propose the following framework which divides the hierarchical community detection into two sub-processes. At the t-th step: **(1)** given a community tree $\mathcal{T}_t$, we partition nodes to communities on the tree; **(2)** given the partitioning results, we further adjust the community tree and obtain $\mathcal{T}_{t+1}$. By alternating the above procedures, we can generate a complete community tree $\mathcal{T}_T$ starting from a root community $\mathcal{T}_0$. The overall framework is shown in Fig. 2, where the generation of the community tree is similar to a growing-up process.

The main advantage of the framework is that we could parametrize the two sub-processes, making it possible apply DNN to implement the framework. In this paper, we propose the following two trainable agents for the above sub-processes respectively, which can be summarized as:

- **Embedding Agent:** We introduce the graph embedding technique to learn node-community affiliation given a community tree. Specifically, we embed both nodes and communities to a hyperbolic space and calculate the similarity between a node and a community according to the metric function defined in the hyperbolic space, which has a natural advantage in modeling hierarchical structure of a graph (Section 3.2).
- **Community Agent:** There are a number of candidate operations which can be used to adjust a community tree. For example, one can derive $\mathcal{T}_{t+1}$ by adding, moving or deleting communities on $\mathcal{T}_t$. In this paper, we mainly consider adding a community at each step because the adding operation is especially suitable for dealing with incoming new observations (Sections 3.3 & 3.4).

3.2 The Embedding Agent

The goal of the embedding agent is to learn node-community affiliation given a community tree $\mathcal{T}_t$ and linkage information Υ. Moreover, it should also describe the hierarchical structure of the graph. In this paper, we introduce the graph embedding technique to achieve the goal.

First, we embed both nodes and communities to the same vector space, and use the similarity between vectors to model node-community affiliation. Specifically, for each node v_i, we use a low-dimensional vector $\boldsymbol{e}_i \in \mathbb{R}^D$ to represent it. Similarly, for a community $c = \{1, \cdots, t\}$ on $\mathcal{T}_t$, we also embed it with a low-dimensional vector $\boldsymbol{e}_c \in \mathbb{R}^D$. For the choice of the vector space, we define it as the Poincaré ball, which is a typical hyperbolic space [28]. Formally, the embeddings should satisfy $\|e\|_2 \leq 1$, and the similarity between a node v_i and a community c can be written as a metric between two embedded vectors:

$$\|\boldsymbol{e}_i - \boldsymbol{e}_c\|_H^2 = \operatorname{arcosh}\Big(1 + \frac{2\|\boldsymbol{e}_i - \boldsymbol{e}_c\|^2}{(1 - \|\boldsymbol{e}_i\|^2)(1 - \|\boldsymbol{e}_c\|^2)}\Big) \tag{1}$$

The key feature of this space is that when the norm of two vectors becomes large, the distance between them increases exponentially. As shown in Fig. 2, the distance between Community 3 and Community 4 will be much larger than the distance between Community 2

and Community 3, which is different from the situation in the Euclidean space. As a result, after the inference, nodes or communities with higher level will have smaller norms and vice versa. Previous work has shown the effectiveness of this technique in hierarchical clustering for images and documents [24]. In this work, we leverage this technique to hierarchical community detection.

Based on the definition, we present how to model node-community affiliation. For each node v_i, we define a Multinomial distribution $P(c|v_i) = \rho_{ic}$ over communities by calculating the similarity between the node and all the communities,

$$\rho_{ic} \propto \exp(-\|\boldsymbol{e}_i - \boldsymbol{e}_c\|_H^2) \tag{2}$$

where $\rho_{ic} \in [0,1]$, $\sum_{c=1}^{t} \rho_{ic} = 1$. The distribution measures the probability of a node belonging to different communities, e.g., in an online social network, a user may be interested on music and sport at the same time. In contrast, heuristic algorithms cannot model such overlapping node-community affiliation [50, 56]. For instance, during the recursive division, each node could only be classified to a certain partition.

Then the remaining problem is how to use $\mathcal{T}_t$ and the link information to learn effective embeddings. An intuitive idea is that for nodes who have no links between other (i.e., $y_{ij} = 0$), their communities should be distant on the tree $\mathcal{T}_t$, and vice versa. Based on this motivation, we propose the following distance function on the community tree $\mathcal{T}_t$. Suppose communities $c_1, c_2 \in \mathcal{T}_t$, and $c_0 \in \mathcal{T}_t$ is the lowest common ancestor of c_1 and c_2. The distance between the two communities on $\mathcal{T}_t$ is defined as $\Lambda(c_1, c_2) = \text{dist}(c_1, c_0) + \text{dist}(c_2, c_0)$, where $\text{dist}(\cdot,\cdot)$ calculates the length of the shortest path between two communities on the tree. Then we can measure the distance between two nodes with the following definition:

$$d_{ij} = \mathbb{E}_{c_i \sim P(c|v_i), c_j \sim P(c|v_j)}\left[\Lambda(c_i, c_j)\right] = \boldsymbol{\rho}_i^T \Lambda \boldsymbol{\rho}_j, \tag{3}$$

where $\Lambda \in \mathbb{R}^{t \times t}$ is the metric matrix of the community tree and $\boldsymbol{\rho}_i \in \mathbb{R}^t$ is the vector representation of ρ_{ic}. In order to minimize d_{ij} for $y_{ij} = 1$ and vice versa, we use the ranking loss:

$$\ell(\phi) = \sum_{v_i} \sum_{v_j, v_k} \max(0, \beta + d_{ik} - d_{ij}), \tag{4}$$

where $y_{ij} = 1$, $y_{ik} = 0$, $\beta > 0$ is a constant margin, and $\phi = \{\boldsymbol{e}_i, \boldsymbol{e}_c\}$ is the set of parameters in node embeddings and community embeddings. However, optimizing $\boldsymbol{e}_i$ and $\boldsymbol{e}_c$ directly is difficult because of the constraint of $\|e\|_2 \leq 1$ in the Poincaré ball. Previous methods use projected gradient descents to solve this problem [28]. In this paper, we alternatively propose to reparametrize the embedding vectors as $\boldsymbol{e}_i = \left(1 - \exp(-\omega(\eta_i))\right) \cdot \frac{\tilde{\boldsymbol{e}}_i}{\|\tilde{\boldsymbol{e}}_i\|^2}$, where $\eta_i \in \mathbb{R}$ is the scale parameter, $\tilde{\boldsymbol{e}}_i$ is the vector in $\mathbb{R}^D$ and $\omega(\cdot)$ is the ReLU function to get non-negative value. Then we can represent embedding $\boldsymbol{e}_i$ by parameters $\tilde{\boldsymbol{e}}_i$ and η_i. In this way, we simplify the original constrained optimization problem to an unconstrained one. Similarly, we can reparametrize the community embeddings $\boldsymbol{e}_c$. Based on this simplification, we can directly apply the standardized gradient-based optimization algorithms to infer parameters $\eta, \tilde{\boldsymbol{e}}$.

3.3 The State Matrix

After optimizing the embedding agent, our framework learns effective overlapping node-community affiliation over $\mathcal{T}_t$. We present below how to further adjust the community tree by inserting new communities. We first concisely discuss under what condition we need to expand the community tree by insertion. From our perspective, a major condition is: the number of existing communities is insufficient to fully model the hierarchical structure of the graph. For instance, for nodes v_i, v_j with no observed links, the embedding model may still cluster v_i, v_j together in the same community. To reduce such misfittings, we need to insert a new community to the existing community tree and separate the nodes to different communities.

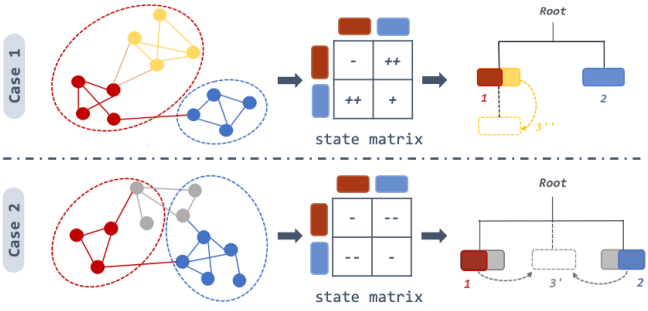

Figure 3: A motivating example for the criterion of community insertion and the design of the state matrix s_t.

To further formulate the condition on insertion, we consider a simple case in Fig. 3. Suppose a community tree $\mathcal{T}_t$ with size $t = 2$ whose two communities are both leaves of the root. With embeddings learned in Sec. 3.2, we can partition nodes into two communities. However, it can be easily found that $t = 2$ is too small to model the network structure. In order to improve the partition, we suggest that the insertion choice should depend on the following criteria:

- **Case 1:** If there exists a group of nodes (colored in orange) which have relation with some nodes in Community 1 and have no relation with nodes in Community 2, in this case, we could insert a new Community $3''$ under Community 1.
- **Case 2:** If there exists a group of nodes (colored in gray) which have few links with nodes in both communities and the embedding model still divides them to two communities, then we should separate these nodes from the two communities and insert a new Community $3'$ under the root.

Based on the above analysis, we propose to define the state information s_t by counting the improperly assigned node-community affiliation relations. Given $\mathcal{T}_t$ and the learned embeddings, we could obtain a matrix s_t of size $t \times t$ to describe the misfitting. The procedural definition of s_t is presented in Alg. 1. Intuitively, if $s_t(c, c)$ is small, it means that there are a large number of node pairs clustered in community c, but they have no co-relations. Similarly, if $s_t(c_1, c_2)$ is small, it means a large number of node pairs with observed links are separated to communities c_1 and c_2.

3.4 The Community Agent

Given the state matrix s_t, the insertion can be described by a policy $\pi(a_t|s_t)$, where $a_t = \{0, 1, \cdots, t\}$ is the community to which we want to insert a new community as its child. Here, 0 indexes the root community. To this end, a straightforward thought is to implement

Algorithm 1 Calculating State Matrix for $\mathcal{T}_t$.

Initialize $s_t \in \mathbb{R}^{t\times t}$ with zeros
for $v_i \in \mathcal{V}$ **do**
  Calculate $c_i = \text{argmax}_c \rho_{ic}$
for $y_{ij} \in \Upsilon$ **do**
  $\tilde{y}_{ij} = 2 \cdot y_{ij} - 1$
  if $c_i = c_j$ **then**
    $s_t(c_i, c_j) = s_t(c_i, c_j) + \tilde{y}_{ij}$
  else
    $s_t(c_i, c_j) = s_t(c_i, c_j) - \tilde{y}_{ij} \cdot \Lambda(c_i, c_j)$
return s_t

the policy through a pre-defined strategy. Nevertheless, designing such strategy is extremely difficult because: **(1)** there may exist more cases in addition to what we have mentioned above; **(2)** when the size of community tree $\mathcal{T}_t$ increases, the state information s_t will become extremely complex, making it difficult to determine how to insert a new community. To tackle this challenge, we propose to implement the policy by a multi-layer neural network, which is proved to be effective in various complex tasks (e.g., [35, 36]). Formally, given the state matrix s_t, we first calculate the hidden layer,

$$h_c = \tanh\big(\sum_{\tau=1}^{t} w_h \cdot s_t(c,\tau) + b_h\big) \tag{5}$$

where $w_h, b_h \in \mathbb{R}$ are parameters in the hidden layer. This layer could reduce the state matrix to a vector representation $h \in \mathbb{R}^t$. Based on the output from the hidden layer, we then predict the position of the inserted community by $o_t = \text{softmax}[w_o^T \cdot h + b_o]$, where $w_o \in \mathbb{R}^{t\times(t+1)}, b_o \in \mathbb{R}^{t+1}$ are parameters in the output layer. We model $o_t \in \mathbb{R}^{t+1}$ as a multinomial distribution over communities. Finally we sample an a_t from the distribution as the predicted position. The parameters of the community agent are denoted as $\theta = \{w_h, b_h, w_o, b_o\}$.

The remaining problem is that the shape of the input and output varies from different step. To tackle the challenge, we use global parameters $w_o \in \mathbb{R}^{T\times(T+1)}$ and $b_o \in \mathbb{R}^{T+1}$. At each step t, we obtain part of the parameters $w_{o,t} = w_o(0:t, 0:t+1)$ and $b_{o,t} = b_o(0:t+1)$.

4 THE REINFORCED FRAMEWORK

4.1 Optimizing the Community Agent

Different from the training of the embedding agent, it is however difficult to optimize the community agent due to the lack of ground-truth labels, i.e., there is no supervision signal about which position a_t is the best choice. In this part, we propose to leverage reinforcement learning (RL) to tackle this challenge. Specifically, we define the following game-playing procedure to train the community agent's policy:

- Randomly initialize the embedding agent;
- For $t = 1, \cdots, T-1$:
  - Update the embedding agent with community tree $\mathcal{T}_t$;
  - Calculate the state matrix s_t with the learned node-community affiliations;
  - Use the community agent $\pi(a|s)$ to build $\mathcal{T}_{t+1}$;
  - Calculate the rewards of the generated community trees $\{\mathcal{T}_t\}$;
  - Update the community agent by maximizing the rewards.

During each round, we first generate a complete community tree $\mathcal{T}_T$, then we estimate the quality of the generation and provide rewards to the community agent. After receiving the signals, the community agent updates its parameters to maximize the rewards. In the next round, it is expected to generate a better community tree. In other words, instead of feeding to the community agent which position a_t is the best choice, we optimize the community agent by indirect labels, that is, by maximizing the quality of the generated tree.

Formally, at each step t, we use the fine-tuned embedding model to output both the state matrix s_t and the reward r_t. The s_t is used to predict a_t to form a new community tree $\mathcal{T}_{t+1}$, and r_t is used to estimate the quality of current tree $\mathcal{T}_t$. The reward r_t is defined as,

$$r_t = \sum_{(v_i,v_k)\in y_{ik}=0} d_{ik} - \sum_{(v_i,v_j)\in y_{ij}=1} d_{ik} \tag{6}$$

where d_{ij} is the distance between nodes v_i, v_j as defined in Eq. 3. A larger r_t means that the model can better cluster similar nodes and better separate distant nodes, vice versa. Based on the definition, in each round of generating a complete community tree, we can sample a sequence of state-action-reward tuples $(s_t, a_t, r_t)_{t=1}^T$, or namely, an *episode*. Then the objective function can be written as,

$$\max_\theta \mathbb{E}_{(s_t,a_t,r_t)_{t=1}^T\sim\pi}\Big[\sum_{t=1}^{T}\gamma^{T-t}\cdot r_t\Big] \tag{7}$$

where γ is a hyper-parameter called the discount factor. In this work, we use the PPO algorithm to infer the parameters, which is the state-of-the-art method towards this problem [33]. Specifically, the PPO algorithm optimizes the following loss function:

$$\ell(\Theta) = \mathbb{E}_{\Lambda_t, a_t\sim\pi}\Big[-\hat{A}_t \cdot \min\Big(R_t(\Theta), \text{clip}\big(R_t(\Theta), 1-\epsilon, 1+\epsilon\big)\Big) + \zeta_1 \cdot A_t^2 - \zeta_2 \cdot q_{\pi(s_t|a_t)}\Big] \tag{8}$$

where ζ_1, ζ_2 are regularization coefficients, $q_{\pi(a_t|s_t)}$ is the entropy of the multinomial distribution on a_t, and:

$$A_t = [\sum_{\tau=t}^{T}\gamma^{\tau-t} r_\tau] - v_t, \;\; , \; R_t(\Theta) = \frac{\pi(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} \tag{9}$$

where $v_t = w_h^T h + b_h$ is the value function to estimate the current reward based on s_t, $w_h \in \mathbb{R}^t, b_h \in \mathbb{R}$ are parameters of the value function v_t, π_{old} is the policy in the last round, and gradients will not go through π_{old} and $\hat{A}_t$. Then we can use algorithm such as Adam [17] to infer $\Theta = \{w_h, w_o, w_v, b_h, b_o, b_v\}$.

During the game-playing process, the embedding agent gives feedback to the community agent and help it learn better community trees. In the meantime, better community trees will help the embedding model learn more effective results. After several rounds the community agent will be able to generate a well-qualified community tree. Besides, we can use the learned embedding results to assign each node a community.

4.2 Online Updating

We further describe the online updating strategy used in our framework. Given a graph with linkage information set Υ_0, we can use the

Algorithm 2 The proposed jointly learning framework.

repeat
  Initialize $\mathcal{T}_1 = \{\text{Root}\}$, $s_1 = [0]$ and $\Lambda_1 = [0]$.
  for $t = 1, \cdots, T$ **do**
    Use the policy $\pi(a_t|s_t)$ to sample a community a_t.
    Insert a new community under a_t.
    Form a new community tree $\mathcal{T}_{t+1}$.
    Calculate the distance matrix Λ_{t+1} of tree $\mathcal{T}_{t+1}$.
    Use $(\Upsilon, \Lambda_{t+1})$ to train embeddings and output $\{\rho_{ic}\}_{c=1}^{t}$.
    Use $(\Upsilon, \Lambda_{t+1}, \{\rho_{ic}\}_{c=1}^{t})$ to calculate the reward r_t.
    Use $(\Upsilon, \Lambda_{t+1}, \{\rho_{ic}\}_{c=1}^{t})$ to calculate state matrix s_{t+1}.
  Use $(s_t, a_t, r_t)_{t=1}^{T}$ to update the parameters θ by PPO.
until Convergence

above framework to learn a community tree and node-community affiliation. When there are new nodes and edges added to the graph (i.e., linkage information set Υ_t), we can use Υ_t and the existing community tree $\mathcal{T}_T$ to perform mini-batch updating for the embedding agent. If the community tree $\mathcal{T}_T$ is not able to describe the new graph structure, e.g., the decreased performance of hierarchical community detection is larger than a pre-defined threshold, we can use the embedding agent to calculate the state matrix s_{T+1}, and insert a new community under a_{T+1} by the community agent. With the new community tree $\mathcal{T}_{T+1}$, we can further fine-tune the community agent to obtain better hierarchical community detection results.

4.3 Parallel Acceleration

To further improve the training, during each game-playing round, we propose to generate L community trees $\{\mathcal{T}_T^{(l)}\}_{l=1}^{L}$ simultaneously, and use these episodes to update the community agent. Compared with only using one episode, multiple episodes help the agent to explore more possible structures in parallel and hence speed up the convergence of learning. Nevertheless, such mechanism will increase the time cost in fact. To tackle the challenge, we design a distributed strategy that scales our algorithm to multiple GPU devices. Specifically, at each step t, we first use the community agent to generate L trees $\{\mathcal{T}_{t+1}^{(l)}\}_{l=1}^{L}$ given $\{\mathcal{T}_t^{(l)}\}_{l=1}^{L}$ in the last step. Then we distribute these trees to workers on multiple GPUs, where each worker learns an embedding agent independently. After the learning, we collect the results $(s_{t+1}^{(l)}, r_t^{(l)})$ and pass them to the community agent to generate trees $\{\mathcal{T}_{t+2}^{(l)}\}_{l=1}^{L}$. Consider that the efficiency bottleneck is the node representation learning as it needs to go through the link set Υ, we use parallelization on GPUs to reduce the computation time of sampling multiple episodes.

4.4 Complexity Analysis

We provide a concise analysis on the complexity of our model here. For each step t, the complexity of the community agent is $O(t^2)$, and the complexity of the embedding agent is $O(|\Upsilon|(t^2 + D))$, where D is the dimension of node embeddings. For T steps, the complexity is $O(|\Upsilon|(T^3 + T \cdot D))$. In our parallelized implementation, the complexity is $O(L \cdot |\Upsilon| \cdot (T^3 + T \cdot D))$, where L is the number of workers in sampling episodes. Compared with the state-of-the-art approaches, the complexity of our original design has a constant scale L larger time cost, while, after the parallelization, the actual running time of our method is the same scale as most of the existing methods. Section 5.5 further compares the empirical time cost.

Table 1: Statistics of datasets.

	Aminer	BlogCatalog	Wiki-Vote	Deezer-RO
Nodes	12840	8943	3513	11847
Edges	190658	660840	95028	105844
Labels	4	39	NA	78
Modularity	✓	✓	✓	✓
NMI	✓	×	×	×
AUC	✓	✓	✓	✓
F1	×	✓	×	✓

5 EMPIRICAL RESULTS

In this section, we validate the effectiveness of our proposed framework. We aim to answer the following research questions:

- **RQ 1:** Is our proposed model able to discover better community tree compared with state-of-the-art methods?
- **RQ 2:** Is our proposed model able to perform effective online updating?
- **RQ 3:** Does our proposed model learn effective node embeddings?

5.1 Experimental Settings

We use four kinds of downstream graph applications in the experiments: graph visualization, clustering analysis, link prediction and multi-label node classification (for simplicity, we refer it as node classification in the following). The former two tasks evaluate the quality of community detection results, and the latter two tasks are designed to validate the effectiveness of the learned node embeddings.

Datasets. We evaluate the effectiveness of our model on 4 graph datasets: Aminer[1], BlogCatalog[2], Wiki-Vote and Deezer[3]. All datasets except Wiki-Vote and Aminer have multiple ground-truth labels for nodes, which can be used for node classification.

Metrics. For clustering analysis, we use modularity and normalized mutual information (NMI) to quantify the community detection results. The modularity is an unsupervised metric, where a larger modularity value means the model can better cluster similar nodes together. The NMI is a supervised metric, which measures the similarity between the community structure and the ground-truth structure. For link prediction, we randomly divide the edges set Υ to a train set and a test set with a ratio of 8:2. Then we use the edges in the train set to infer parameters in the model, and predict the probability of links in the test set by node embeddings. We use Area Under the relative operating Characteristic (AUC) [5], which is widely used in the performance evaluation of binary classification tasks, to measure whether the embeddings capture the correct graph structure. For multi-label node classification, each

[1] https://www.aminer.cn/billboard/aminernetwork
[2] http://socialcomputing.asu.edu/datasets/BlogCatalog3
[3] https://snap.stanford.edu/data

Table 2: Results for community detection (Modularity) and link prediction (AUC), where - means the results are unavailable, e.g., Louvain does not have node embeddings, and N/A means the model fails to converge in 2 days.

	Modularity				AUC				NMI
	Aminer	Wiki-Vote	BlogCatalog	Deezer-RO	Aminer	Wiki-Vote	BlogCatalog	Deezer-RO	Aminer
LINE	-	-	-	-	0.583	0.691	0.526	0.500	-
GNE	-	-	-	-	0.670	0.666	0.565	0.582	-
GEMSEC	0.661	0.211	0.021	0.649	0.941	0.742	0.506	0.498	0.361
Louvain	0.647	0.307	0.159	0.603	-	-	-	-	0.539
HCDE	0.689	0.210	0.180	0.037	-	-	-	-	0.410
MNMF	0.709	0.297	0.154	0.665	0.950	0.843	0.725	**0.923**	0.294
vGraph	0.710	0.258	N/A	N/A	0.824	0.685	N/A	N/A	0.001
ComE	0.745	0.309	0.139	0.740	0.505	0.530	0.699	0.912	0.765
ReinCom	**0.759**	**0.403**	**0.224**	**0.742**	**0.960**	**0.884**	**0.848**	0.912	**0.798**

node has several ground-truth labels. We split nodes to a train set and a test set of rate 8:2. Then we train a multi-layer neural network as the classifier. Given a node, the classifier learns to output the probability that the node belongs to each categories of different labels. Finally, we predict the probabilities for the test set and use the F1 score to measure the accuracy.

Baselines. We choose 6 state-of-the-art community detection methods as the baselines. Specifically, we compare our methods with 4 non-hierarchical methods: Graph Embedding with Self Clustering (GEMSEC) [32], vGraph [40], Modularity Nonnegative Mattrix Factorization (MNMF) [46] and Community Embedding (ComE) [6]. These non-hierarchical community detection methods also leverage the graph embedding technique, which is proved to be effective in various applications. We also compare our model with two hierarchical methods. The first is the Louvain algorithm [25], which leverages a division-based heuristic strategy to search for a community tree which maximizes the modularity. This method is proved to be effective among various traditional approaches such as the label propagation algorithm (LPA). The second is the Hierarchical Community Detection by Embeddings (HCDE) [38], which proposes to perform hierarchical clustering on learned node embeddings. Furthermore, in order to validate the quality of our learned node embeddings, we compare our methods with 2 node representation learning baseline methods: Large-scale Information Network Embedding (LINE) [41] and Galaxy Network Embedding (GNE) [10], where the GNE first finds a community tree by Louvain algorithm and learns node embeddings given the tree. Finally, we also conduct several self-comparison to validate the design of our framework.

Others. For more implementation details of our model, we use Adam [17] as the gradient optimizer, with the learning rate 0.003 and the regularization coefficient 0.5. For all community detection methods, we set $T = 15$ for the Wiki-Vote dataset, and $T = 20$ for the others. The reason is the that Wiki-Vote dataset is smaller than the others. Since Louvain and HCDE use a dendrogram to represent the community tree and are not able to output a fixed number of communities as a result, we extract their learned communities by a threshold of depth 4. For the dimension of node embeddings, we choose 50 for our method and use the optimal setting reported in the original papers of the baselines. Besides:

Table 3: F1 value for node classification.

	Deezer-RO		BlogCatalog	
	Macro-F1	Micro-F1	Macro-F1	Micro-F1
LINE	0.023	0.302	0.062	0.190
GNE	0.029	0.396	0.016	0.071
ComE	0.029	0.314	0.018	0.053
GEMSEC	0.023	0.277	0.107	0.263
ReinCom	**0.058**	**0.401**	**0.138**	**0.281**

- Rather than starting from $\mathcal{T}_0$, we initialize the tree with $\mathcal{T}_{T_0}$, where there are T_0 leaves under the root community. In our experiment we set $T_0 = 4$.
- As for γ in calculating the discounted accumulative rewards, we have tried $\gamma = 0.6, 0.8, 0.95, 0.99$, among which $\gamma = 0.95$ is proved to be the optimal choice.
- For the number of workers in our parallel implementation, a larger L usually provides better results. In this paper we set $L = 90$ based on our computation resources.
- For the choice of y_{ij}, we use the same setup as in Node2vec.

For predictive applications, we use 10-fold cross validation. All the experiments are conducted on a machine with two Intel Xeon 4210 CPUs and six NVIDIA RTX 2080Ti GPUs.

5.2 The Results for RQ 1

5.2.1 Main Results. We first validate the effectiveness of hierarchical modeling in community detection. As we can see from the comparison between hierarchical method such as Louvain and non-hierarchical method GEMSEC, the hierarchical modeling leads to better community detection results. For instance, on Wiki-Vote, the modularity of Louvain is 0.307, while the one of GEMSEC is only 0.211. Besides, our model achieves noticeable improvements on most datasets compared with previous hierarchical methods. Especially on Wiki-Vote, our modularity is 0.403, which is about 33% relatively higher than that of the best baseline. Furthermore, we take a deeper look at the hierarchical structure discovered by our model and previous hierarchical methods. We define the metric

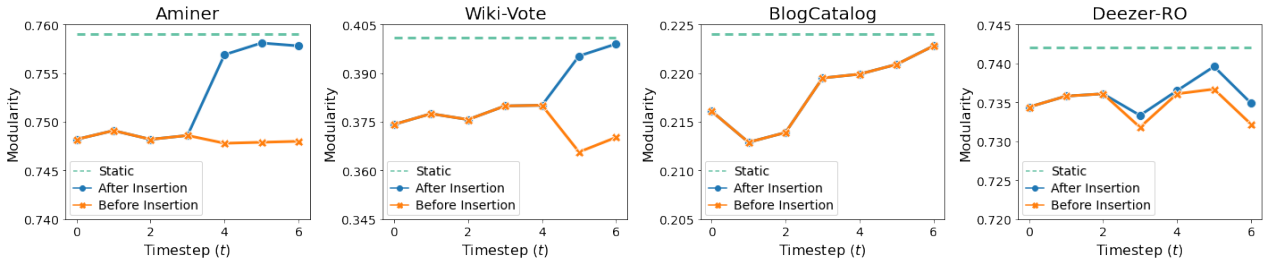

Figure 4: The curves of modeling effectiveness in terms of modularity during the online updating process.

Table 4: Comparisons of the average tree distance between nodes w. and w/o. a link between one another.

	Aminer		Wiki-Vote		Deezer-RO	
	$\bar{d}_+$	$\bar{d}_-$	$\bar{d}_+$	$\bar{d}_-$	$\bar{d}_+$	$\bar{d}_-$
Louvain	9.119	9.135	7.794	7.959	5.904	5.905
HCDE	5.803	5.809	5.120	5.121	6.858	6.864
ReinCom	0.631	3.707	0.821	2.191	0.949	1.873

Table 5: Self-comparisons on two datasets.

	Aminer		Wiki-Vote	
	Modularity	AUC	Modularity	AUC
Non-hierarchical	0.703	0.950	0.326	0.853
Random	0.719	0.957	0.295	0.846
w/o. Hyperbolic	0.728	0.948	0.295	0.870
ReinCom	**0.759**	**0.960**	**0.327**	**0.884**

as $\bar{d}_+ = \frac{1}{|\Upsilon_+|}\sum_{(i,j)\in y_{ij}=1}\Lambda(c_i,c_j)$, where $\Lambda(c_i,c_j)$ is the distance of two communities on the community tree defined in Section 3.2. Similarly, we can define $\bar{d}_-$ for those $y_{ij}=0$. A smaller $\bar{d}_+$ means that the model could cluster similar nodes together, and a larger $\bar{d}_-$ means that the model could separate dissimilar nodes. As we can see from Table 4, for previous hierarchical methods, the negative distance $\bar{d}_-$ is close to the positive distance $\bar{d}_+$. In contrast, our model distinguishes $y_{ij}=1$ and $y_{ij}=0$ well. On Aminer, for the node pairs which have a link between one another, the averaged distances on the community tree is 0.631, while the distance becomes 3.707 if there is no link between them.

We speculate that the improved modeling effectiveness of ReinCom mainly comes from the following reasons. On one side, our community agent could discover a better hierarchical structure of the graph. On the other side, our embedding agent could learn better node-community affiliation based on the community tree. To validate this, we conduct the following experiments. First, we replace the learned community tree $\mathcal{T}_T$ by a non-hierarchical tree and a random tree respectively. Then we train the embedding agent with these two community trees and report the results. As we can see from Table 5, the performance is greatly reduced after using incorrect community trees. The second self-comparison is to replace the hyperbolic space of the embedding agent with a Euclidean space. As we can see from Table 5, the modularity also decreases on both datasets after this modification. In summary, the capability of ReinCom in modeling hierarchical community structures and overlapping node-community affiliation is a key factor to its improved performance.

5.2.2 The Influence of Overlapping Modeling. We further investigate the influence of overlapping modeling. As shown in Table 2, for those non-hierarchical methods, the overlapping modeling could help the model learn better node-community affiliation. For instance, the MNMF and vGraph outperform GEMSEC on most datasets. Furthermore, an interesting finding is that overlapping modeling is more important than hierarchical modeling on certain graphs. For instance, on Aminer, the ComE outperforms hierarchical community detection methods Louvain and HCDE. As the first hierarchical method which is able to model overlapping node-community affiliation, ReinCom outperforms these baselines on all datasets.

5.3 The Results for RQ 2

We validate our proposed online updating strategy on four datasets. We divide the datasets into several parts. For $t=0$, we construct Υ_0 by 91% nodes and their edges. For $t=1$ to $t=6$, we incrementally add 1.5% nodes and their edges to the graph at each timestamp. When the model receive Υ_t, it first updates the embedding agent with these incremental data. If the drop of the modularity is greater than a threshold, we use the community agent to insert a new community to build $\mathcal{T}_{t+1}$ and further fine-tune the embedding agent. The results are shown in Fig. 4. The orange line is the performance of conducting mini-batch updating on the embedding agent without updating the community tree, and the blue line is the performance of inserting a new community when the drop of the modularity is greater than a threshold. The green line is the result of training both agents on the whole data (from $t=0$ to $t=6$). As we can see, the utility of our model still remains high after receiving new nodes and edges. Furthermore, after inserting a new community, ReinCom can even improve the modeling effectiveness at the previous timestamp. For instance, on Aminer and Wiki-Vote, the result is close to training the model on the whole dataset. We also validate the change of the existing partitions, where we find that previous node-community affiliation only slightly changes after the online updating. The above experimental results strongly justify the effectiveness of our proposed online updating strategy.

5.4 The Results for RQ 3

For RQ 3, we validate the effectiveness of our model in three graph applications: link prediction, node classification and graph visualization. According to the link prediction results in Table 2, our method outperforms all baselines in AUC by a large margin on

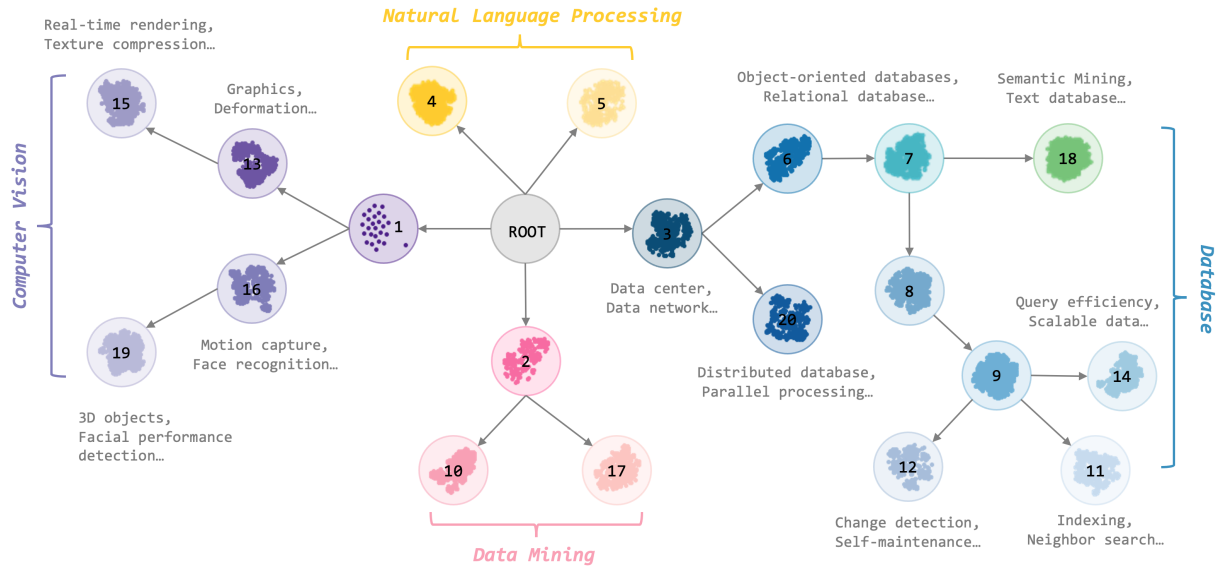

Figure 5: Visualization of the hierarchical architecture of the Aminer network learned by our proposed ReinCom.

most datasets. For example, on BlogCatalog, the result of our model is 0.848, while the best baseline is 0.725. For node classification, our method also consistently shows better performance over all datasets. We do not provide the results of vGraph and MNMF in this scenario because they fail to converge in 2 days.

Finally, we present an interesting graph visualization method based on ReinCom. Specifically, we first visualize the community tree. Then for each community on the tree, we conduct t-SNE [22] with embeddings $\boldsymbol{e}_i$ that belong to the community, and scale the coordinates. We visualize the learned community tree and the node embeddings after dimension reduction on Aminer in Fig. 5. To explain, there are four kinds of papers in the Aminer citation network: data mining, computer vision, natural language processing and database. As we can see from Fig. 5, our model separates different kinds of papers well without knowing the ground-truth label. Besides, there are several interesting findings. First, if the number of database papers becomes much larger, our model could capture this feature and model these nodes with larger sub-trees. Second, nodes belong to deeper communities have more specific topics. For instance, papers belong to Cluster 13 often discuss general problems in computer vision such as graphics and deformation, while papers belong to Cluster 15 often discuss more specific topics such as real-time rendering. Third, our model captures the overlapping node-community affiliation. For instance, some papers in Cluster 18 also belong to Clusters 4 & 5 because they discuss topics such as semantic mining and text database, which are also relevant to natural language processing.

5.5 More Results

Additionally, we present the running time of our methods. As analyzed in Section 4.4, the time complexity of our method is constant times to size of Υ. The results in Table 6 empirically prove the theoretical time complexity. We compare our methods with two non-hierarchical community detection methods MNMF and vGraph, which are also graph embedding based approaches. The complexity of MNMF and vGraph are constant times to $N \times N$ and Υ respectively. With the help of distributed training strategy, our methods could be even faster than some non-hierarchical counterparts. For instance, in Deezer-HR dataset, vGraph and MNMF fail to model this dataset due to the intractable time cost, while our method converges within hours.

Table 6: Inference time of different methods.

	Wiki-Vote	Deezer-RO	Deezer-HR
Nodes	3513	11847	42586
Edges	95028	105844	935138
MNMF	4min	39min	N/A
vGraph	240min	N/A	N/A
ReinCom	45min	50min	500min

6 CONCLUSION

In this paper, we present the first deep learning based framework on hierarchical community detection. With the aid of deep learning techniques, our proposed framework, ReinCom, shows its advantages not only in discovering hierarchical structures, but also in enabling online updating and facilitating various downstream graph-based tasks including link prediction, node classification and visualization. Empirical results on four real-world complex networks validate the aforementioned strengths of ReinCom. For future works, it would be interesting to integrate more candidate operations on the community tree into the community agent. Besides, as this work is the first step in leveraging deep learning techniques for hierarchical community detection, we mainly conduct proof-of-concept evaluation. Although the proposed ReinCom is by-design able to be applied to networks with millions of nodes, as is discussed in the time complexity analysis in Section 4.4, it may need extra optimization efforts to cater for industry-scale networks such as the Twitter social network, which is left as a future work.

ACKNOWLEDGMENTS

This work was supported in part by National Natural Science Foundation of China (61972099, U1836213,U1836210,U1736208), and Natural Science Foundation of Shanghai (19ZR1404800). Min Yang is a faculty of Shanghai Institute of Intelligent Electronics & Systems, Shanghai Institute for Advanced Communication and Data Science, and Engineering Research Center of CyberSecurity Auditing and Monitoring, Ministry of Education, China.

REFERENCES

[1] Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. 2008. Mixed membership stochastic blockmodels. *JMLR* (2008).
[2] Sivaraman Balakrishnan, Min Xu, Akshay Krishnamurthy, and Aarti Singh. 2011. Noise thresholds for spectral clustering. In *NeurIPS*.
[3] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.
[4] Charles Blundell and Yee Whye Teh. 2013. Bayesian hierarchical community discovery. In *NeurIPS*.
[5] Kendrick Boyd, Kevin H Eng, and C David Page. 2013. Area under the precision-recall curve: point estimates and confidence intervals. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 451–466.
[6] Sandro Cavallari, Vincent W Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. 2017. Learning community embedding with community detection and node embedding on graphs. In *CIKM*.
[7] Aaron Clauset, Cristopher Moore, and Mark EJ Newman. 2006. Structural inference of hierarchies in networks. In *ICML Workshop on Statistical Network Analysis*. Springer, 1–13.
[8] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27.
[9] Anirban Dasgupta, John Hopcroft, Ravi Kannan, and Pradipta Mitra. 2006. Spectral clustering by recursive partitioning. In *European Symposium on Algorithms*.
[10] Lun Du, Zhicong Lu, Yun Wang, Guojie Song, Yiming Wang, and Wei Chen. 2018. Galaxy Network Embedding: A Hierarchical Community Structure Preserving Approach.. In *IJCAI*. 2079–2085.
[11] Illés Farkas, Dániel Ábel, Gergely Palla, and Tamás Vicsek. 2007. Weighted network modules. *New Journal of Physics* 9, 6 (2007), 180.
[12] Santo Fortunato. [n.d.]. Community detection in graphs. *Physics reports* ([n. d.]).
[13] Sara E Garza and Satu Elisa Schaeffer. 2019. Community detection with the label propagation algorithm: a survey. *Physica A: Statistical Mechanics and its Applications* (2019), 122058.
[14] Michelle Girvan and Mark EJ Newman. [n.d.]. Community structure in social and biological networks. *PNAS* ([n. d.]).
[15] Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. 2017. Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach. *arXiv preprint arXiv:1706.05674* (2017).
[16] Alexandre Hollocou, Thomas Bonald, and Marc Lelarge. 2016. Improving PageRank for local community detection. *arXiv preprint arXiv:1610.08722* (2016).
[17] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[18] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[19] Tianxi Li, Lihua Lei, Sharmodeep Bhattacharyya, Koen Van den Berge, Purnamrita Sarkar, Peter J Bicke, and Elizaveta Levina. [n.d.]. Hierarchical community detection by recursive partitioning. *J. Amer. Statist. Assoc.* ([n. d.]).
[20] Fanzhen Liu, Shan Xue, Jia Wu, Chuan Zhou, Wenbin Hu, Cecile Paris, Surya Nepal, Jian Yang, and Philip S Yu. 2020. Deep learning for community detection: progress, challenges and opportunities. *arXiv preprint arXiv:2005.08225* (2020).
[21] Vince Lyzinski, Minh Tang, Avanti Athreya, Youngser Park, and Carey E Priebe. 2016. Community detection and classification in hierarchical stochastic blockmodels. *IEEE Transactions on Network Science and Engineering* 4, 1 (2016), 13–26.
[22] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.
[23] Fragkiskos D Malliaros and Michalis Vazirgiannis. [n.d.]. Clustering and community detection in directed networks: A survey. *Physics reports* ([n. d.]).
[24] Nicholas Monath, Manzil Zaheer, Daniel Silva, Andrew McCallum, and Amr Ahmed. 2019. Gradient-based Hierarchical Clustering using Continuous Representations of Trees in Hyperbolic Space. In *KDD*.
[25] Mark EJ Newman and Michelle Girvan. 2004. Finding and evaluating community structure in networks. *Physical review E* 69, 2 (2004), 026113.
[26] Mark EJ Newman and Elizabeth A Leicht. 2007. Mixture models and exploratory analysis in networks. *PNAS* (2007).
[27] Andrew Y Ng, Michael I Jordan, and Yair Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *NeurIPS*.
[28] Maximillian Nickel and Douwe Kiela. 2017. Poincaré embeddings for learning hierarchical representations. In *NeurIPS*.
[29] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *nature* 435, 7043 (2005), 814–818.
[30] Giulio Rossetti and Rémy Cazabet. 2018. Community discovery in dynamic networks: a survey. *ACM Computing Surveys (CSUR)* 51, 2 (2018), 1–37.
[31] Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2019. Linear-time Hierarchical Community Detection. *arXiv preprint arXiv:1906.06432* (2019).
[32] Benedek Rozemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. 2019. Gemsec: Graph embedding with self clustering. In *IEEE/ACM ASONAM*.
[33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint* (2017).
[34] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *TPAMI* (2000).
[35] David Silver, Aja Huang, Chris J Maddison, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.
[36] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.
[37] Jiří Šíma and Satu Elisa Schaeffer. 2006. On the NP-completeness of some graph cluster measures. In *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer, 530–537.
[38] Blaž Škrlj, Jan Kralj, and Nada Lavrač. 2019. Embedding-based Silhouette Community Detection. *arXiv preprint arXiv:1908.02556* (2019).
[39] Tom AB Snijders and Krzysztof Nowicki. 1997. Estimation and prediction for stochastic blockmodels for graphs with latent block structure. *Journal of classification* (1997).
[40] Fan-Yun Sun, Meng Qu, Jordan Hoffmann, Chin-Wei Huang, and Jian Tang. 2019. vGraph: A Generative Model for Joint Community Detection and Node Representation Learning. In *NeurIPS*. 512–522.
[41] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *WWW*.
[42] Lei Tang and Huan Liu. 2011. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery* 23, 3 (2011), 447–478.
[43] Joshua R Tyler, Dennis M Wilkinson, and Bernardo A Huberman. 2005. E-mail as spectroscopy: Automated discovery of community structure within organizations. *The Information Society* 21, 2 (2005), 143–153.
[44] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).
[45] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. 2019. Knowledge graph convolutional networks for recommender systems. In *WWW*.
[46] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community preserving network embedding. In *AAAI*.
[47] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *SIGIR*.
[48] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. 2020. Disentangled Graph Collaborative Filtering *(SIGIR)*.
[49] Bryce G Westlake and Martin Bouchard. 2016. Liking and hyperlinking: Community detection in online child sexual exploitation networks. *Social science research* 59 (2016), 23–36.
[50] Jierui Xie, Stephen Kelley, and Boleslaw K Szymanski. 2013. Overlapping community detection in networks: The state-of-the-art and comparative study. (2013).
[51] Jierui Xie, Boleslaw K Szymanski, and Xiaoming Liu. 2011. Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In *IEEE international conference on data mining workshops*.
[52] Dongkuan Xu and Yingjie Tian. 2015. A comprehensive survey of clustering algorithms. *Annals of Data Science* 2, 2 (2015), 165–193.
[53] Jaewon Yang and Jure Leskovec. 2012. Community-affiliation graph model for overlapping network community detection. In *ICDM*.
[54] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*.
[55] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*. 4800–4810.
[56] Wayne W Zachary. 1977. An information flow model for conflict and fission in small groups. *Journal of anthropological research* 33, 4 (1977), 452–473.
[57] Shihua Zhang, Rui-Sheng Wang, and Xiang-Sun Zhang. 2007. Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A: Statistical Mechanics and its Applications* 374, 1 (2007).
[58] Xiaojin Zhu and Zoubin Ghahramani. 2002. Learning from labeled and unlabeled data with label propagation. (2002).