CLARE: A Semi-supervised Community Detection Algorithm

Xixi Wu
21210240043@m.fudan.edu.cn
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
China

Yun Xiong*
yunx@fudan.edu.cn
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University, Peng Cheng Laboratory, Shenzhen
China

Yao Zhang
yaozhang@fudan.edu.cn
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
China

Yizhu Jiao
yizhuj2@illinois.edu
University of Illinois at Urbana-Champaign
United States

Caihua Shan
caihuashan@microsoft.com
Microsoft Research Asia
China

Yiheng Sun
elisun@tencent.com
Tencent Weixin Group
China

Yangyong Zhu
yyzhu@fudan.edu.cn
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
China

Philip S. Yu
psyu@uic.edu
University of Illinois at Chicago
United States

ABSTRACT

Community detection refers to the task of discovering closely related subgraphs to understand the networks. However, traditional community detection algorithms fail to pinpoint a particular kind of community. This limits its applicability in real-world networks, e.g., distinguishing fraud groups from normal ones in transaction networks. Recently, semi-supervised community detection emerges as a solution. It aims to seek other similar communities in the network with few labeled communities as training data. Existing works can be regarded as seed-based: *locate seed nodes and then develop communities around seeds*. However, these methods are quite sensitive to the quality of selected seeds since communities generated around a mis-detected seed may be irrelevant. Besides, they have individual issues, e.g., inflexibility and high computational overhead. To address these issues, we propose **CLARE**, which consists of two key components, **C**ommunity **L**ocator **a**nd Community **Re**writer. Our idea is that we can *locate potential communities and then refine them*. Therefore, the community locator is proposed for quickly locating potential communities by seeking subgraphs that are similar to training ones in the network. To further adjust these located communities, we devise the community rewriter. Enhanced by deep reinforcement learning, it suggests intelligent decisions, such as adding or dropping nodes, to refine community structures flexibly. Extensive experiments verify both the effectiveness and efficiency of our work compared with prior state-of-the-art approaches on multiple real-world datasets.

CCS CONCEPTS

• **Computing methodologies** → **Reinforcement learning**; • **Information systems** → **Clustering**.

KEYWORDS

semi-supervised community detection, subgraph matching, reinforcement learning

ACM Reference Format:
Xixi Wu, Yun Xiong, Yao Zhang, Yizhu Jiao, Caihua Shan, Yiheng Sun, Yangyong Zhu, and Philip S. Yu. 2022. CLARE: A Semi-supervised Community Detection Algorithm. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539370

1 INTRODUCTION

Networks are a powerful framework to represent rich relational information among data objects from social, natural, and academic domains [2, 25]. A crucial step to understand a network is identifying and analyzing closely related subgraphs, i.e., communities. The research task of discovering such subgraphs from networks is known as the community detection problem [39], which can reveal the structural patterns and inherent properties of networks.

However, traditional community detection algorithms are incapable of pinpointing a particular kind of community. In some scenarios, there are various types of communities in the same network, while people may only focus on a specific type, i.e., the

*Corresponding author

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539370

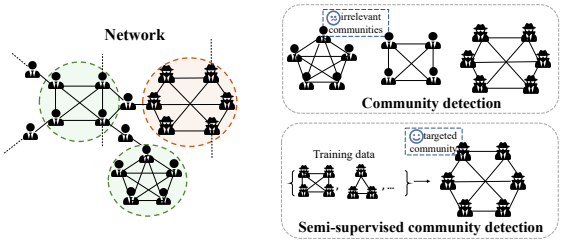


Figure 1: Tasks comparison. This is a subgraph of a trading network with two normal communities (green circles) and one fraud community (red circle). Traditional community detection methods may identify both kinds of communities. However, semi-supervised community detection methods can utilize some fraud groups to pinpoint the remaining fraud community.

targeted community. Traditional community detection methods cannot handle these situations, as they rely solely on overall structural information for inference [5, 9, 36], failing to capture the inherent features of some certain targeted communities. For example, as shown in Figure 1, they cannot distinguish fraud groups from normal ones in transaction networks, and instead exhaustedly identify both kinds of communities.

Therefore, some researchers turn to semi-supervised settings to identify targeted communities. Formally, semi-supervised community detection is defined as utilizing certain communities as training data to recognize the other similar communities in the network. With such training data, it can implicitly learn the distinct features of the targeted community [41]. Semi-supervised community detection is a promising research problem with a wide range of real-world applications. For example, it can detect social spammer circles in social networks to avoid malicious attacks [13].

There are existing works on semi-supervised community detection, e.g., Bespoke [4] and SEAL [41]. As shown in Figure 2, they can be generalized as seed-based methods: *first locate prospective seed nodes (central nodes) in the network and then develop communities around seeds*. However, these methods are quite sensitive to the quality of selected seeds since communities generated around a mis-detected seed may be irrelevant. Besides, these methods have individual issues. For instance, Bespoke restricts community structures to 1-ego net, i.e., a central node along with its neighbors. This makes Bespoke inflexible as failing to generalize to communities with arbitrary structures. SEAL alleviates this issue by introducing generative adversarial networks (GAN). It formulates the generation of a community into sequential decisions process, as the generator of GAN would add one node in each step. However, it suffers from high computational overhead since developing a community in such way is quite expensive.

To solve the aforementioned challenges, we infer communities from a novel subgraph perspective: *first locate potential communities and then refine them*. Specifically, we leverage the training communities to locate similar subgraphs via matching, and further adjust them based on their structural context. The benefits are threefold: (1) A subgraph carries more structural patterns as well as inherent features than a single seed node, promising to bring more precise

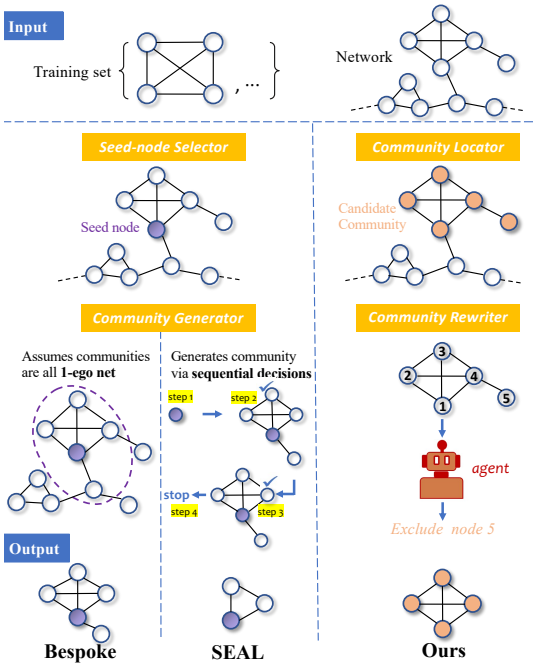


Figure 2: An illustration of Bespoke (left), SEAL (middle), and our proposed CLARE (right). Bespoke and SEAL first locate seed nodes and then develop communities around seeds. On the contrary, we propose a different solution: first locate potential communities and further rewrite them.

positioning. (2) Predicting communities directly from a subgraph scale can avoid the expensive cost of generating communities from scratch. (3) With the introduction of global structural patterns, the refining process can further optimize located communities.

Inspired by the above insights, we propose a novel semi-supervised community detection framework, **CLARE**. It consists of two key components, **C**ommunity **L**ocator **a**nd Community **Re**writer. The community locator can quickly locate potential communities by seeking subgraphs that are similar to training ones. Specifically, we encode communities into vectors, measure the similarities between communities in the latent space, and then discover candidates based on the similarities with the nearest neighbors matching strategy. The community rewriter is proposed to further adjust those candidate communities by introducing global structural patterns. We frame such refinement process as a deep reinforcement learning task and optimize this process via policy gradient. For located communities, the rewriter provides two actions: adding outer nodes or dropping existing nodes, thus refining their structures flexibly and intelligently.

We summarize the contributions of this work as follows:

- We study the semi-supervised community detection problem from a novel subgraph perspective. Different from existing seed-based methods, our solution can be regarded as first locating potential communities and then refining them.
- We propose a novel framework CLARE, consisting of Community Locator and Community Rewriter. The community locator is

proposed to quickly locate potential communities by seeking subgraphs similar to training ones. And the community rewriter can further fine-tune those located communities. Enhanced by deep reinforcement learning, it provides actions such as adding or dropping nodes, refining community structures flexibly.
- We conduct experiments on real-world networks as well as hybrid networks containing various types of communities. Compared with both community detection and semi-supervised community detection baselines, our model achieves outstanding performance and remains competitive on efficiency. Moreover, our model exhibits robustness even when encountering data sparsity or network noises.

2 RELATED WORK

2.1 Community Detection

A common definition of community detection is to partition graph nodes into multiple groups, where internal nodes are more similar than the external. Some works [5, 9, 29] are optimization-based methods that search a graph partition by optimizing some metrics such as modularity. Moreover, there are matrix factorization based methods [21, 33] which learn latent representations for communities by decomposing adjacency matrices. Studies like [36, 37] are generative models that infer communities by fitting the original graph. In recent years, some frameworks that combine graph representation learning and community detection have been proposed [7, 14, 30, 40]. For example, vGraph [30] is a probabilistic generative model to learn community membership and node representation collaboratively. In a word, these community detection works fail to pinpoint a particular kind of community.

Semi-supervised community detection. This task aims to seek the rest communities in the network with few labeled communities as training data. Existing methods [4, 41] usually pick seed nodes in the network by leveraging training data and then develop communities around seeds. However, they are quite sensitive to the quality of selected seeds. Besides, they have individual issues, e.g., inflexibility [4] and high computational overhead [41]. We skip work [16] because their semi-supervised setting is completely different from current discussion.

2.2 Subgraph Matching

Subgraph matching refers to the task of determining the existence of the query graph in the target graph [38]. We will discuss the development of subgraph matching methods as the implementation of community locator is inspired by subgraph matching. Conventional algorithms such as [32] only focus on graph structures. Other works like [1, 10] also consider categorical node features.

Neural network based methods. As graph neural networks (GNN) raise a surge of interest [12, 19, 34], GNN-based matching methods have been proposed [3, 11, 20, 38] and have achieved great results. A recent breakthrough in this domain is [38] which significantly outperforms other subgraph matching baselines thanks to the introduction of order embedding. Inspired by their work, we design Community Order Embedding to capture the similarities between communities and further match candidates. Especially, our novelty lies in migrating the matching method into detection tasks.

2.3 Graph Combinatorial Optimization with RL

With the success of deep reinforcement learning in games [24], researchers have attempted to utilize reinforcement learning (RL) techniques for the graph combinatorial optimization (GCO) problem [18, 22, 27, 41]. S2V-DQN [18] is the first attempt that models a GCO problem into a sequential decision problem with deep Q-learning. And in [22] the authors propose Graph Pointer Networks to solve TSP problems. SEAL [41] adopts policy gradient to learn heuristics for generating a community. Recently, Shan et al. [27] propose a RL-based framework named RG-Explainer for generating explanations for graph neural networks. Note that rewriting a community, i.e., adjusting its members, is also a GCO problem. Thus, in this paper, we resort to RL for optimizing the community rewriter.

3 METHODOLOGY

Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges, and $\mathbf{X}$ is the node feature matrix. With m labeled communities $\dot{\mathcal{C}} = \{\dot{C}^1, \dot{C}^2, ..., \dot{C}^m\} (\forall_{i=1}^{m} \dot{C}^i \subset G)$ as training data, our goal is to find the set of other similar communities $\hat{\mathcal{C}}$ ($|\hat{\mathcal{C}}| \gg |\dot{\mathcal{C}}|$) in G. The important notations used are listed in Table 1.

Table 1: Important Notations

Notations	Definition
$G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$	Network
$\dot{\mathcal{C}} = \{\dot{C}^1, ..., \dot{C}^m\}$	The set of training communities
$\mathcal{N}(u)$	Neighbors of node u
$\mathbf{x}(u)$	Raw features of node u
$\mathbf{x}'(u)$	Augmented features of node u
$\mathbf{z}(u)$	Embedding of node u
$\mathbf{z}(C^i)$	Embedding of community C^i
$\partial C = \cup_{u \in C} \mathcal{N}(u) \setminus C$	Outer boundary of the community C
$\mathbf{s}_t(u)$	Representation of node u at time t

As shown in Figure 3, we first describe the inference process of CLARE. It consists of 2 components, **Community Locator** and **Community Rewriter**. For a specified training community, the locator would seek its best match from all candidate communities. For efficiency, k-ego net (a central node along with its neighbors within k hops) of each node in the network is regarded as a candidate. Considering that the community detected by locator may not be accurate, as its structure is fixed, we introduce Community Rewriter for further refinement. It acts like an intelligent agent that adjusts the community structures by performing actions. Specifically, it determines whether to exclude existing nodes or expand towards the outer boundary (the neighbors of nodes in this community) during each step. This iterative process is repeated until encountering "STOP" signal, at which point we obtain the final predicted community.

3.1 Community Locator

In this subsection, we explain our community locator component. Specifically, we implement community order embedding to encode each community and match candidate communities. In this way, we can quickly locate potential communities.

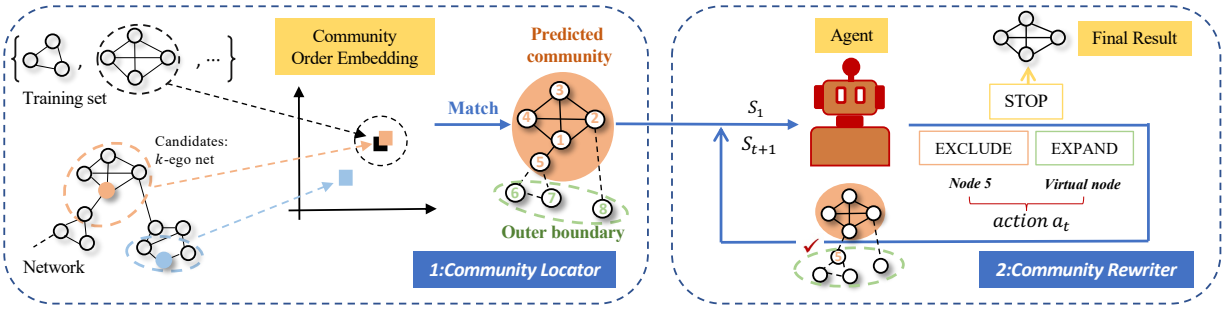


Figure 3: CLARE inference framework. For a specified training community, suppose our goal is to detect a new community similar to it in the network. We utilize community locator for locating the best-matched candidate community. We further rewrite this candidate community by feeding it to a well-trained structure agent. After performing incremental "EXCLUDE" and "EXPAND" actions suggested by this agent, we obtain the final predicted community.

3.1.1 Design goal. We map each community into a d-dimensional vector and ensure that the subgraph relationship can be properly reflected in the embedding space: if community C^a is a subgraph of community C^b, then corresponding community embedding $\mathbf{z}(C^a)$ has to be in the "lower-left" of embedding $\mathbf{z}(C^b)$:

$$\mathbf{z}(C^a)[i] \leq \mathbf{z}(C^b)[i],\ \forall_{i=1}^{d}, \text{iff } C^a \subseteq C^b.$$

We adopt this idea of order embedding proposed in [38] to implement community order embedding. To our knowledge, migrating the matching method into detection tasks is quite novel. When the embeddings of two communities are very close to each other, we consider these two communities as isomorphic. To realize this goal, the max margin loss is utilized for training:

$$\mathcal{L} = \sum_{Pos} E\left(\mathbf{z}(C^a), \mathbf{z}(C^b)\right) + \sum_{Neg} \max\left\{0, \alpha - E\left(\mathbf{z}(C^a), \mathbf{z}(C^b)\right)\right\},$$

$$\text{where } E\left(\mathbf{z}(C^a), \mathbf{z}(C^b)\right) = \left\|\max\left\{0,\ \mathbf{z}(C^a) - \mathbf{z}(C^b)\right\}\right\|_2^2. \quad (1)$$

Here Pos denotes the set of positive samples where community C^a is a subgraph of C^b in each pair (C^a, C^b), Neg denotes the set of negative samples, and margin α is a distance parameter.

3.1.2 Identifying potential communities. We utilize the well-trained community order embedding for identifying potential communities. In order to quickly divide the entire network into small subgraphs, we regard the k-ego net of each node in the network as a candidate community. As we feed these candidates to the encoder, we can get their representations $\mathbf{Z} = \{\mathbf{z}(C^1), \ldots, \mathbf{z}(C^{|\mathcal{V}|})\}$ where C^i denotes the k-ego net of node $i \in \mathcal{V}$. Similarly, training communities are encoded as $\dot{\mathbf{Z}} = \{\mathbf{z}(\dot{C}^1), \ldots, \mathbf{z}(\dot{C}^m)\}$, each of which is treated as a pattern for matching. Suppose our goal is to predict N new communities, then the n ($n = \frac{N}{m}$) candidate communities closest to each training one in the embedding space are considered as predicted results.

We also have an alternative method for detecting arbitrary numbers of communities in the lack of prior knowledge N. Specifically, we can use some threshold η to take any community whose similarity to the closest training community is above η. Since the identified communities are ranked by the similarity measures, i.e., the distance in the latent space, we can stop when the identified community is not that similar to what we are looking for. In this way, the requirement of N can be avoided.

3.1.3 Graph Neural Networks based encoder. Here we describe the concrete encoding process of communities. For a specific node $u \in \mathcal{V}$, we augment its feature as $\mathbf{x}'(u) = [\mathbf{x}(u), degree(u), \max(DN(u)), \min(DN(u)), \text{mean}(DN(u)), \text{std}(DN(u))]$ where $\mathbf{x}(u)$ is the raw features of node u with default value as 1 and $DN(u) = \{degree(v) | v \in \mathcal{N}(u)\}$. Hence, the expressive power of node attributes can be boosted, which is crucial for networks without node features [6, 41]. We adopt graph neural networks (GNN) [12, 19, 34] as the basic component of the encoder Z.

Firstly, the original node representations undergo a linear transformation as $\mathbf{z}^{(0)}(u) = \mathbf{W}^1 \cdot \mathbf{x}'(u)$. Then, the encoder propagates and aggregates the information via a **k-layers GNN**. We study the impact of different graph neural networks in the experiments and will discuss later. Note that we ensure the number of layers is in consistent with the choice of ego net dimension k. Finally, we concatenate the node embeddings in previous layers and perform linear transformation again to obtain the final embedding $\mathbf{z}(u)$ of node u as follows:

$$\mathbf{z}(u) = \mathbf{W}^2 \cdot \textbf{CONCAT}\left(\mathbf{z}^{(0)}(u), \ldots, \mathbf{z}^{(k)}(u)\right),$$

where $\mathbf{z}^{(l)}(u)$ denotes the embedding of node u in the l-th layer and $\mathbf{W}^j (j = 1, 2)$ are trainable parameters. For a specific community C^i, its embedding is calculated as $\mathbf{z}(C^i) = \sum_{u \in C^i} \mathbf{z}(u)$.

So far, the optimization process is summarized in Algorithm 1.

3.2 Community Rewriter

In Community Locator, we make an assumption on the structure of predicted communities for efficiency while sacrificing flexibility. Therefore, Community Rewriter is further proposed to provide flexibility via intelligent refinement.

The core idea is that, given a raw predicted community C, we can either drop some irrelevant nodes in C or absorb nodes from its outer boundary, i.e., $\partial C = \cup_{u \in C} \mathcal{N}(u) \setminus C$, to obtain a more accurate estimation. Such structural refinement can be considered as a combinatorial optimization problem, where we need to pick suitable nodes in a set to optimize the objective. Therefore, we adopt

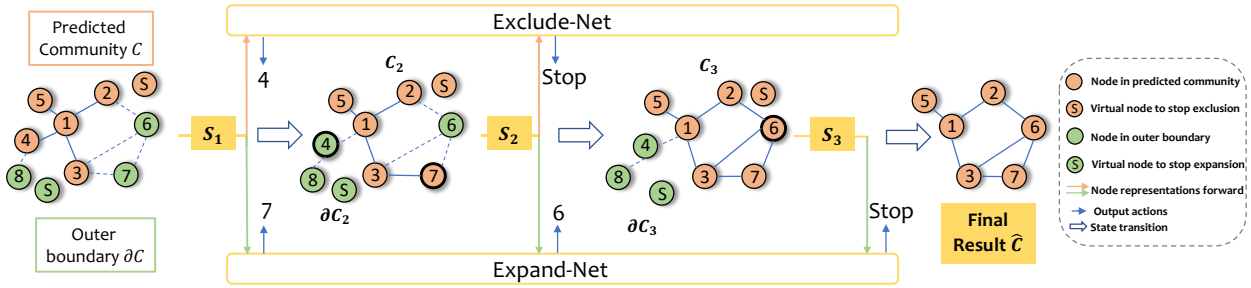


Figure 4: Community Rewriting Process. For a community C detected in the first stage, we associate it with its outer boundary ∂C. At the t-th step, Exclude-Net would pick a node in C_t for exclusion. Similarly, Expand-Net selects a node from ∂C_t for expansion. The next state is moved to after performing both Exclude and Expand actions. If both Exclude-Net and Expand-Net have decided to "STOP", the refinement ends and final predicted community is generated.

RL for automatically learning heuristics to rewrite communities. An illustrative example of rewriting process is shown in Figure 4.

3.2.1 Task design. The refinement of a community detected by the locator is regarded as an episode. Specifically, the initial state is defined as $S_1 = C \cup \partial C$ where C is a raw predicted community and ∂C is its boundary. At the t-th step, we have the current predicted community C_t and its boundary ∂C_t. We further pick a node a_t^{exclude} in C_t for exclusion as well as a node a_t^{expand} from ∂C_t for expansion. We would modify the community as $C_{t+1} = C_t \setminus \{a_t^{\text{exclude}}\} \cup \{a_t^{\text{expand}}\}$. In this way, we move to the next state. It is worth noting that the state is the combined representations of both C_t and ∂C_t. The action space for exclusion is C_t while ∂C_t serves for expansion. We also assign a reward value for rewriting.

State. As mentioned before, the state at timestamp t is denoted as $S_t = C_t \cup \partial C_t$. For any node $u \in S_t$, we give it a node representation $\mathbf{s}_t(u)$. We design the initial node representation $\mathbf{s}_1(u)$ by augmenting its first-stage embedding $\mathbf{z}(u)$ with the information of raw community C:

$$\mathbf{s}_1(u) = \textbf{CONCAT}\left(\mathbf{z}(u), \mathbb{I}\{u \in C\}\right),$$

where $\mathbb{I}\{\text{condition}\}$ is the indicator function that returns 1 if the condition is true, otherwise 0.

Each node could further combine information from its current neighborhood. To achieve this, we also utilize a GNN-based network D parameterized by θ. Thus, at the t-th step, the representation of node u is updated as:

$$D_\theta(\mathbf{s}_t(u)) = \textbf{GNN}(\mathbf{s}_t(u), \{\mathbf{s}_t(v) | v \in S_t\}),$$

$$\mathbf{s}_{t+1}(u) = \textbf{CONCAT}\left(D_\theta(\mathbf{s}_t(u)), \mathbb{I}\{u \in C_{t+1}\}\right).$$

Action. During each step, we consider adding a new node and dropping an existing node simultaneously. In other words, the action suggested at the t-th step is composed of both a_t^{exclude} and a_t^{expand}. Specifically, we take C_t as the action space for "EXCLUDE" while ∂C_t serves for "EXPAND".

We design respective policy networks for these two kinds of actions. They share similarities in the overall architecture: basic Multilayer Perceptron (MLP) [26] ends with a softmax layer to measure the probability of taking a specified node:

$$a_t^{\text{exclude}} = U_\phi(C_t) = \textbf{Softmax}\left(\textbf{MLP}_\phi(\{\mathbf{s}_t(u) | u \in C_t\})\right),$$

$$a_t^{\text{expand}} = P_\psi(\partial C_t) = \textbf{Softmax}\left(\textbf{MLP}_\psi(\{\mathbf{s}_t(v) | v \in \partial C_t\})\right),$$

where ϕ is the trainable parameter of Exclude-Net U and ψ is the trainable parameter of Expand-Net P.

We add a special "STOP" action into the action space to learn stopping criteria. Concretely, we construct a **virtual node** to represent the stop signal. It is an isolated node with all-zero features. If Exclude-Net selects the virtual node, we will stop excluding and merely expand towards the outer boundary. Similarly, we do not expand when the virtual node is chosen for expansion. If both exclusion and expansion end, so does the refinement process.

Reward. Since our rewriting goal is to obtain a more accurate community estimation via incremental actions, we directly take one of the evaluation metrics for reward computation, i.e., F1 score. Given the suggested action a_t at the t-th step, we define the reward r_t at that time as the difference of F1 score brought by a_t.

3.2.2 Optimization. We learn the rewriting agent $A = \{\phi, \psi, \theta\}$ via **policy gradient** [17, 31]. Since those three parameters share the similar process of being updated, we take the Exclude-Net U parameterized by ϕ as an example to illustrate the optimization.

Given a community C detected in the first stage, we can form an exclusion trajectory $\tau = \{S_1, a_1, S_2, a_2, ..., S_T, a_T, S_{T+1}\}$ (we omit the superscript "exclude" for brevity here). Then the reward r_t obtained by performing the Exclude action a_t is computed as:

$$r_t = \delta(C_{t+1}, \dot{C}^i) - \delta(C_t, \dot{C}^i), \tag{2}$$

where δ denotes F1 score and $\dot{C}^i$ denotes the corresponding ground-truth community. Following the same way, we further calculate rewards for all Exclude actions $a_t(\forall_{t=1}^T)$ in τ. Finally, we update Exclude-Net$_\phi$ according to policy gradient:

$$\phi \leftarrow \phi + lr \sum_{t=1}^{T} \nabla_\phi \log U_\phi(a_t | S_t) \cdot r_t, \tag{3}$$

where lr stands for the learning rate of optimizing Exclude-Net U parameterized by ϕ.

To realize the above optimization objective, a lot of training samples are constructed. Specifically, the generation of a sample follows this way: firstly, we randomly pick a node u from a training community $\dot{C}^i$. Then, its k-ego net C^u as well as corresponding boundary ∂C^u are extracted. Repeatedly, the set of training samples is constructed as $\mathcal{D} = \{(C^u \cup \partial C^u, \dot{C}^i)\}$ where C^u is a k-ego net with

u coming from some training community $\dot{C}^i \in \dot{\mathcal{C}}$. Notably, we fix the structure of these training samples as k-**ego net** to simulate the communities detected by the community locator and $\dot{C}^i$ is utilized for calculating the rewards.

For a training sample $(C^u \cup \partial C^u, \dot{C}^i)$, the agent will produce a trajectory $\tau = \{(S_t, a_t, r_t, S_{t+1})\}$ where a_t is composed of a_t^{exclude} and a_t^{expand}. We calculate the reward r_t associated with $\dot{C}^i$ according to Equation 2. The next state S_{t+1} is generated by taking the action a_t^{expand} and a_t^{exclude} from the current state S_t. When the Expand or Exclude action selects the virtual node, this kind of action will stop. If both expansion and exclusion are stopped, the episode ends and we obtain a complete trajectory τ. Based on the collected trajectory τ for each training sample, we update parameters in the agent. The detailed process is described in Algorithm 2.

3.3 Summary

We summarize the CLARE framework as shown in Algorithm 3. Recall that with m labeled communities as training data, our goal is to detect N new communities in the network.

Firstly, we train the community locator by leveraging known communities. Then we take each training community as a pattern for matching n closest candidate communities in the embedding space ($n = \frac{N}{m}$). Actually, the k-ego net of each node in the network serves as a candidate. After matching, we can get N raw predicted communities. Next, we train the community rewriter. For each community detected in the first stage, it is fed to this well-trained agent and refined into a new community. Finally, we obtain N modified communities as final results.

4 EXPERIMENTS

In this section, we conduct extensive experiments to verify both the effectiveness and efficiency of CLARE on multiple datasets. We also compare the robustness of existing semi-supervised community detection algorithms under different conditions. Due to the space limitation, we move the implementation details and parameters study to Appendix.

4.1 Experiment Setup

Evaluation metrics. For networks with ground-truth communities, the most used evaluation metrics are bi-matching **F1 score** and **Jaccard score** [4, 8, 14, 41]. Given M ground truth communities $\{\dot{C}^j\}$ and N generated communities $\{\hat{C}^i\}$, we compute scores as:

$$\frac{1}{2}\left(\frac{1}{N}\sum_i \max_j \delta(\hat{C}^i, \dot{C}^j) + \frac{1}{M}\sum_j \max_i \delta(\hat{C}^i, \dot{C}^j)\right), \tag{4}$$

where δ can be F1 or Jaccard function. Besides, we use the overlapping normalized mutual information (**ONMI**) [23] as a supplementary metric, which is the overlapping-version of NMI score. For more information of ONMI, please refer to [23].

Datasets. To comprehensively assess the effectiveness of our model, we conduct experiments both on **single datasets** (a network with partially labeled communities) and **hybrid datasets** (combination of multiple different single datasets).

Table 2: Statistics of datasets. C_{Max} denotes the largest community size while C_{Avg} denotes the average community size.

	#N	#E	#C	C_{Max}	C_{Avg}
Amazon	6,926	17,893	1,000	30	9.38
DBLP	37,020	149,501	1,000	16	8.37
Livejournal	69,860	911,179	1,000	30	13.00
Amazon+DBLP	43,946	172,394	2,000	30	8.88
DBLP+Livejournal	106,880	1,070,680	2,000	30	10.69

Single datasets. We choose three common real-world networks containing overlapping communities from SNAP[1] (Amazon, DBLP, and Livejournal). Note that these networks are partially labeled, i.e., most nodes do not belong to any community. Thus, we can view that there are other types of communities in the networks, and our targeted communities are the labeled ones.

Hybrid datasets. We create hybrid networks by combining two different networks following related works [41]. For example, we stack Amazon and DBLP by randomly adding 5,000 cross-networks links between two graphs, resulting in a bigger network. Since communities in different networks exhibit different features [35], we obtain a single network with various types of communities in this way. Similarly, we combine DBLP and Livejournal by adding 10,000 cross-networks links. These two networks are named "Amazon+DBLP" and "DBLP+Livejournal", respectively. Note that we only add some cross-networks links, so the internal connectivity between communities will not be disturbed.

The statistics of all datasets are shown in Table 2. For each single dataset, we use 90 communities as the training set, 10 as the validation set, and the rest as the test set. As to hybrid datasets, we aim to pinpoint one kind of community. For example, on Amazon+DBLP, we would take 90 communities from Amazon for training, 10 for validation, and the remaining communities from Amazon serve for testing.

Compared baselines. We compare CLARE with the following methods: (1) BigClam [36] and (2) its assisted version BigClam-A, (3) ComE [7], (4) CommunityGAN [14], (5) vGraph [30], (6) Bespoke [4] and (7) SEAL [41]. Methods (1)-(5) are strong community detection baselines while (6)-(7) are semi-supervised methods requiring training communities. Other simple baselines like GCN+K-Means have been shown inferior performance [7, 40], and thus we skip those methods. We limit the numbers of their outputs to **1000** communities. For methods (1)-(5), we filter detected communities who have more than 50% overlaps with communities in training/validation sets as SEAL [41] does. The data pre-processing steps and comparing methods are detailed in Appendix A.1 and A.2, respectively.

4.2 Overall Performance

Experimental results are shown in Table 3. For hybrid dataset "Amazon+DBLP", we conduct experiments that utilize 100 communities from DBLP as prior knowledge to detect the remaining DBLP communities in the combined network, as well as utilizing 100 communities from Amazon to detect the remaining Amazon communities.

[1] http://snap.stanford.edu/data/

Table 3: Summary of the performance in comparison with baselines. N/A means the model fails to converge in 2 days. We report the results of CLARE with k=1 on DBLP while k=2 on all other datasets.

	Dataset	BigClam	BigClam-A	ComE	CommunityGAN	vGraph	Bespoke	SEAL	CLARE
F1	Amazon	0.6885	0.6562	0.6569	0.6701	0.6895	0.5193	0.7252	**0.7730**
	DBLP	0.3217	0.3242	N/A	0.3541	0.1134	0.2956	0.2914	**0.3835**
	Livejournal	0.3917	0.3934	N/A	0.4067	0.0429	0.1706	0.4638	**0.4950**
	Amazon*DBLP	0.1759	0.1745	N/A	0.0204	0.0769	0.0641	0.2733	**0.3988**
	DBLP*Amazon	0.2363	0.2346	N/A	0.0764	0.1002	0.2464	0.1317	**0.2901**
	DBLP*Livejournal	0.0909	0.0859	N/A	0.0251	0.0131	0.0817	0.1906	**0.2480**
	Livejournal*DBLP	0.2183	0.2139	N/A	0.0142	0.0206	0.1893	0.2291	**0.2894**
Jaccard	Amazon	0.5874	0.5623	0.5691	0.6045	0.5721	0.4415	0.6792	**0.6827**
	DBLP	0.2186	0.2203	N/A	0.2830	0.0645	0.2593	0.2143	**0.3132**
	Livejournal	0.3102	0.3076	N/A	0.3183	0.0222	0.1324	0.3795	**0.4027**
	Amazon*DBLP	0.1102	0.1095	N/A	0.0109	0.0421	0.0488	0.2419	**0.3241**
	DBLP*Amazon	0.1485	0.1478	N/A	0.0610	0.0555	0.2135	0.0879	**0.2166**
	DBLP*Livejournal	0.0523	0.0485	N/A	0.0120	0.0066	0.0756	0.1485	**0.1893**
	Livejournal*DBLP	0.1505	0.1464	N/A	0.0097	0.0105	0.1503	0.1907	**0.2308**
ONMI	Amazon	0.5865	0.5625	0.5570	0.6040	0.5532	0.4129	0.6862	**0.7015**
	DBLP	0.1113	0.1110	N/A	0.2324	0.0020	0.2347	0.1603	**0.2600**
	Livejournal	0.2696	0.2641	N/A	0.3171	<1e-4	0.1024	0.3695	**0.3703**
	Amazon*DBLP	0.0305	0.0334	N/A	<1e-4	< 1e-4	0.0364	0.2475	**0.3126**
	DBLP*Amazon	0.0471	0.0477	N/A	0.0523	<1e-4	**0.1780**	0.0380	0.1566
	DBLP*Livejournal	0.0113	0.0065	N/A	<1e-4	<1e-4	0.0723	0.1155	**0.1331**
	Livejournal*DBLP	0.0858	0.0795	N/A	0.0053	<1e-4	0.1248	0.1906	**0.2012**

They are denoted as DBLP*Amazon and Amazon*DBLP, respectively. Similarly, we conduct experiments named DBLP*Livejournal and Livejournal*DBLP. From Table 3, we find that:

- CLARE achieves noticeable improvements on almost all datasets compared with all the baselines, demonstrating its exceptional performance. The improvements on hybrid datasets are more significant, indicating its superiority in pinpointing the targeted community.
- Community detection algorithms are shown their unsuitability in targeting a specific kind of community, as they perform poorly on hybrid datasets. For example, CommunityGAN [14] is the best baseline model on DBLP while its performance degrades dramatically on all hybrid datasets. CommunityGAN learns node-community membership matrix and assigns each node into some community. vGraph [30] also assumes that each node belongs to multiple communities. These approaches are more like clustering all nodes in the network rather than locating some specific type of communities. On hybrid datasets, assigning total 106,880 nodes into 1000 clusters could generate irrelevant communities on extremely large scale, resulting in inferior performance.
- Semi-supervised community detection models (CLARE, SEAL [41], and Bespoke [4]) gain better predicted results on both kinds of datasets generally. For example, SEAL is the best baseline model on most datasets.
- Bespoke performs well on DBLP, because the community structures on this dataset is closest to its assumption, 1-ego net. When other datasets do not conform to this structural assumption, the performance degrades. This also exposes the inflexibility and poor generalization of Bespoke.

4.3 Ablation Study

To evaluate the effectiveness of both components of CLARE, we conduct ablation experiments. The ONMI results are shown in Table 4. Due to space limitation, we omit the results of F1 score and Jaccard score, which show similar trends with ONMI.

Community Locator. We report the communities detected by the community locator as "Locator" in Table 4. For an intuitive comparison, we also generate the same number of random subgraphs in the form of k-ego net. We can see that the locator already obtains excellent performance as the improvement compared with random subgraphs is significant. Notably, solely locator has outperformed most baselines, showing the effectiveness of locating targeted communities from the matching approach.

Community Rewriter. By comparing the results between "Locator" and "CLARE", it is clear that the introduction of rewriter can obtain better performance. The improvements on DBLP related datasets are relatively marginal. This is because the community structures on DBLP are very close to 1-ego net form, it only takes few actions to rewrite.

We conduct a case study to see how rewriter works as shown in Figure 5. It learns quite different heuristics for different networks, demonstrating its adaptability and flexibility. For example, on Amazon, many nodes in the same ground-truth community are not detected in the first stage, but the rewriter can intelligently absorb them. Besides, on Livejournal, though raw predicted results include few irrelevant nodes (not in the same ground-truth community with others), many of them can be correctly eliminated.

Table 4: Ablation Study on ONMI score. "k-ego subgraph" is generated by randomly selecting some subgraphs in the form of k-ego net. "Locator" denotes raw communities detected by the community locator. Note that "CLARE" are obtained via rewriting those communities detected by the locator.

	Amazon	DBLP	Livejournal	Amazon*DBLP	DBLP*Amazon	DBLP*Livejournal	Livejournal*DBLP
k-ego subgraph	0.4323	0.1112	0.1140	0.0632	0.0855	0.0365	0.0726
Locator	0.6586	0.2585	0.3592	0.3088	0.1546	0.1322	0.1964
CLARE	**0.7015**	**0.2600**	**0.3703**	**0.3126**	**0.1566**	**0.1331**	**0.2012**

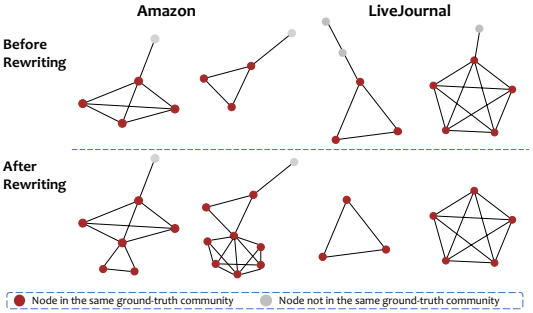


Figure 5: Case study of the community rewriter. On Amazon, many undetected nodes can be correctly absorbed while irrelevant nodes are correctly removed on Livejournal.

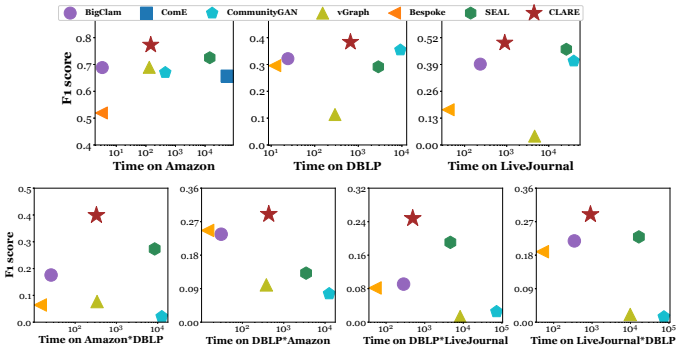


Figure 6: Efficiency comparison with all baselines.

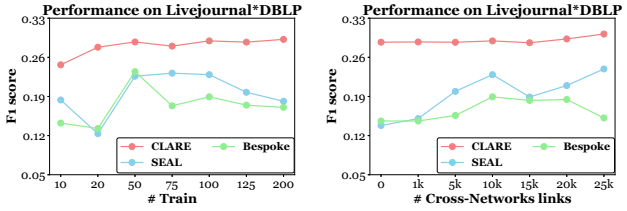
4.4 Efficiency Study

We evaluate the efficiency of CLARE by directly comparing the total running time (training plus testing) with all baselines. In this evaluation, all parameters for baselines are set following their original papers. Figure 6 illustrates the performance (F1 score) and running time (second). Since ComE fails to converge in two days on all datasets except Amazon, we do not show it on those datasets.

Notably, the running time of CLARE is consistently competitive. Even on the largest dataset with totally 106,800 nodes, it takes CLARE only about 1000 seconds. Simultaneously, its performance (F1 score) beats that of other quicker models.

4.5 Discussions

In this section, we compare the robustness of existing semi-supervised community detection algorithms under different conditions: (1) different numbers of training communities; (2) different levels of network noises, i.e., different numbers of cross-networks links.



(a) Number of training communities (b) Number of cross-networks links

Figure 7: The performance trend of semi-supervised community detection algorithms under different conditions.

4.5.1 Different numbers of training communities. With the number of training communities (Livejournal) ranging from 10 to 200 progressively, we compare the performance of Bespoke, SEAL, and CLARE on Livejournal*DBLP. As shown in Figure 7(a), we find that:

- CLARE can learn from training data progressively. With the number of training communities increasing, the performance of CLARE firstly undergoes improvement and then remains stable. Actually, richer subgraph patterns and rewriting patterns can be sampled from increasing training communities, resulting in more accurate results.
- On the contrary, Bespoke and SEAL show fluctuations on performance. This is because their performance relies heavily on the seed selector, which is vulnerable to limited training data.

4.5.2 Different numbers of cross-networks links. Recall that we create hybrid datasets via adding cross-networks links. We view different numbers of added links as different levels of network noises. Because with the number of added links increasing, there will be some new densely connected subgraphs, promising to be mis-detected. We set the number of cross-networks links ranging from 0 to 25,000 and report the results in Figure 7(b). We can see CLARE is quite robust while SEAL and Bespoke are rather volatile. This also indicates that these seed-based methods are susceptible to interference from network noises.

4.6 Application on the attributed graphs

Considering that our experimental networks are non-attributed graphs, we supplement an experiment to show that our model can also achieve good performance on the attributed graphs. We use

the Facebook dataset in SEAL [41], consisting of 3,622 nodes, 76,596 edges, 317 node attributes, and 130 communities. We follow the same experimental settings with SEAL.

From Table 5, we can see that CLARE outperforms SEAL even without node attributes and performs much better when attributes are available. It indicates that CLARE not only captures the structural features of training communities but also effectively utilizes node attributes to obtain more accurate predicted results.

Table 5: Experimental results on attributed network Facebook. CLARE w/o attr. means CLARE without attributes.

	SEAL	CLARE w/o attr.	CLARE	**Improv**
F1	0.3402	0.3829	**0.4126**	21.28%
Jaccard	0.2491	0.2815	**0.3047**	22.32%

5 CONCLUSION

In this paper, we study the semi-supervised community detection problem from a new subgraph perspective. We propose CLARE where the community locator can quickly locate communities and the community rewriter further refines their structures. Specifically, we formulate the structural refinement as a graph combinatorial optimization based on RL. Experiments on real-world datasets prove both the effectiveness and efficiency of our proposal. As for future work, we will try other RL-based optimization methods to further improve the effectiveness of rewriting.

ACKNOWLEDGMENTS

This work is funded in part by the National Natural Science Foundation of China Projects No. U1936213 and No. U1636207. This work is also supported in part by NSF under grants III-1763325, III-1909323, III-2106758, and SaTC-1930941.

REFERENCES

[1] Boanerges Aleman-Meza, Christian Halaschek-Wiener, Satya Sanket Sahoo, A. Sheth, and Ismailcem Budak Arpinar. 2005. Template Based Semantic Similarity for Security Applications. In *ISI*.
[2] Lars Backstrom, Daniel P. Huttenlocher, Jon M. Kleinberg, and Xiangyang Lan. 2006. Group formation in large social networks: membership, growth, and evolution. In *KDD*.
[3] Yunsheng Bai, Haoyang Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. 2019. SimGNN: A Neural Network Approach to Fast Graph Similarity Computation. In *WSDM*.
[4] Arjun Bakshi, Srinivasan Parthasarathy, and Kannan Srinivasan. 2018. Semi-Supervised Community Detection Using Structure and Size. In *ICDM*. 869–874.
[5] 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 (2008), 10008.
[6] Chen Cai and Yusu Wang. 2019. A simple yet effective baseline for non-attributed graph classification. arXiv:1811.03508 [cs.LG]
[7] Sandro Cavallari, V. Zheng, HongYun Cai, K. Chang, and E. Cambria. 2017. Learning Community Embedding with Community Detection and Node Embedding on Graphs. In *CIKM*.
[8] Tanmoy Chakraborty, Ayushi Dalmia, Animesh Mukherjee, and Niloy Ganguly. 2016. Metrics for Community Analysis: A Survey. arXiv:1604.03512 [cs.SI]
[9] Aaron Clauset, Mark E. J. Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Physical review. E, Statistical, nonlinear, and soft matter physics* 70 6 Pt 2 (2004).
[10] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (sub) graph isomorphism algorithm for matching large graphs. In *PAMI*.
[11] Michelle Guo, Edward Chou, De-An Huang, Shuran Song, Serena Yeung, and Li Fei-Fei. 2018. Neural Graph Matching Networks for Fewshot 3D Action Recognition. In *ECCV*.
[12] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.
[13] Xia Hu, Jiliang Tang, and Huan Liu. 2014. Online Social Spammer Detection. In *AAAI*.
[14] Yuting Jia, Qinqin Zhang, Weinan Zhang, and Xinbing Wang. 2019. CommunityGAN: Community Detection with Generative Adversarial Nets. In *WWW*.
[15] Yizhu Jiao, Yun Xiong, Jiawei Zhang, Yao Zhang, Tianqi Zhang, and Yangyong Zhu. 2020. Sub-graph Contrast for Scalable Self-Supervised Graph Representation Learning. In *ICDM*.
[16] Di Jin, Ziyang Liu, Weihao Li, Dongxiao He, and Weixiong Zhang. 2019. Graph Convolutional Networks Meet Markov Random Fields: Semi-Supervised Community Detection in Attribute Networks. In *AAAI*.
[17] Sham M. Kakade. 2001. A Natural Policy Gradient. In *NIPS*.
[18] Elias Boutros Khalil, Hanjun Dai, Yuyu Zhang, Bistra N. Dilkina, and Le Song. 2017. Learning Combinatorial Optimization Algorithms over Graphs. In *NIPS*.
[19] Thomas Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. *ArXiv* abs/1609.02907 (2017).
[20] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. 2019. Graph Matching Networks for Learning the Similarity of Graph Structured Objects. In *ICML*.
[21] Ye Li, Chaofeng Sha, Xin Huang, and Yanchun Zhang. 2018. Community Detection in Attributed Graphs: An Embedding Approach. In *AAAI*.
[22] Qiang Ma, Suwen Ge, Danyang He, Darshan D. Thaker, and Iddo Drori. 2019. Combinatorial Optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning. *ArXiv* abs/1911.04936 (2019).
[23] Aaron F. McDaid, Derek Greene, and Neil Hurley. 2013. Normalized Mutual Information to evaluate overlapping community finding algorithms. arXiv:1110.2515 [physics.soc-ph]
[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518 (2015), 529–533.
[25] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. 2014. Focused clustering and outlier detection in large attributed graphs. In *KDD*.
[26] Frank Rosenblatt. 1963. PRINCIPLES OF NEURODYNAMICS. PERCEPTRONS AND THE THEORY OF BRAIN MECHANISMS. *American Journal of Psychology* 76 (1963), 705.
[27] Caihua Shan, Yifei Shen, Yao Zhang, Xiang Li, and Dongsheng Li. 2021. Reinforcement Learning Enhanced Explainer for Graph Neural Networks. In *NIPS*.
[28] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *ArXiv* abs/1811.05868 (2018).
[29] Jianbo Shi and Jitendra Malik. 1997. Normalized cuts and image segmentation. In *CVPR*. 731–737.
[30] 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 *NIPS*.
[31] Richard S. Sutton, David A. McAllester, Satinder Singh, and Y. Mansour. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *NIPS*.
[32] Julian R. Ullmann. 1976. An Algorithm for Subgraph Isomorphism. *Journal of the ACM (JACM)* 23 (1976), 31 – 42.
[33] Xiao Wang, Di Jin, Xiaochun Cao, Liang Yang, and Weixiong Zhang. 2016. Semantic Community Identification in Large Attribute Networks. In *AAAI*.
[34] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks? *ArXiv* abs/1810.00826 (2019).
[35] Jaewon Yang and Jure Leskovec. 2012. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems* 42 (2012), 181–213.
[36] Jaewon Yang and Jure Leskovec. 2013. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *WSDM*. 587–596.
[37] Jaewon Yang, Julian McAuley, and Jure Leskovec. 2013. Community Detection in Networks with Node Attributes. In *ICDM*.
[38] Rex Ying, Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. 2020. Neural Subgraph Matching. arXiv:2007.03092 [cs.LG]
[39] Hongyi Zhang, Irwin King, and Michael R. Lyu. 2015. Incorporating Implicit Link Preference Into Overlapping Community Detection. In *AAAI*.
[40] Tianqi Zhang, Yun Xiong, Jiawei Zhang, Yao Zhang, Yizhu Jiao, and Yangyong Zhu. 2020. CommDGI: Community Detection Oriented Deep Graph Infomax. In *CIKM*.
[41] Yao Zhang, Yun Xiong, Yun Ye, Tengfei Liu, Weiqiang Wang, Yangyong Zhu, and Philip S. Yu. 2020. SEAL: Learning Heuristics for Community Detection with Generative Adversarial Networks. In *KDD*. 1103–1113.

A REPRODUCIBILITY

We release CLARE at https://github.com/FDUDSDE/KDD2022CLARE. We implement CLARE in Pytorch[2], PyG[3], and DeepSNAP[4]. All experiments are conducted on a single NVIDIA Tesla V100 SXM2 with 32G memory.

A.1 Data Pre-processing

We use the networks with ground-truth communities (Amazon, DBLP, and Livejournal) provided by SNAP. For conducting experiments, we perform the following pre-processing:

(1) We omit communities whose size are beyond the 90-th percentile. For example, the largest community in DBLP contains 7,556 nodes, while the 90-th percentile is only 16. By doing so, we can exclude outliers.

(2) Furthermore, we randomly select 1000 communities from these retrieved ones. This number is a trade-off between maintaining a relatively large network and being scalable on most baselines. Note that ComE [7], CommunityGAN [14], and vGraph [30] mainly use networks with thousands of nodes in their original papers. They can hardly execute on networks consisting of total communities due to huge memory utilization.

(3) For each dataset, we extract a subgraph that contains only the nodes in communities and their corresponding outer boundaries. In this way, we obtain the final datasets for experiments.

For hybrid datasets, given that Amazon and Livejournal are significantly different in size (almost 1:10 in scale), we skip merging these two datasets and just consider Amazon+DBLP and DBLP+Livejournal combinations.

A.2 Comparing methods

During experiments, we consider both community detection and semi-supervised community detection strong baselines.

Community detection algorithms:

- **BigClam** [36]: This is a strong baseline for overlapping community detection based on matrix factorization. We also consider an assisted version of BigClam following the work [4], denoted by **BigClam-A**.
- **ComE** [7]: This is a framework that jointly optimizes community embedding, community detection, and node embedding.
- **CommunityGAN** [14]: This is a method that extends the generative model of BigClam from edge level to motif level.
- **vGraph** [30]: This is a probabilistic generative model to learn community membership and node representation collaboratively.

Semi-supervised community detection algorithms:

- **Bespoke** [4]: This is a semi-supervised community detection algorithm based on structure and size information.
- **SEAL** [41]: This is the start-of-the-art semi-supervised community detection algorithm that aims to learn heuristics for community detection based on GAN.

Executable file for BigClam is from SNAP. Codes for ComE, CommunityGAN, vGraph, Bespoke, SEAL are provided by the authors.

ComE [7] can not be converged within 2 days on most datasets, so we report N/A. CommunityGAN [14] and vGraph [30] suffer from high memory utilization. So we employ the mini-batch strategy with the batch size of 5000 or 15000 for optimizing vGraph. As to CommunityGAN, we replace its Adam optimizer with a SGD optimizer for reducing memory usage. In addition, we all follow their default parameters settings.

A.3 Implementation Details

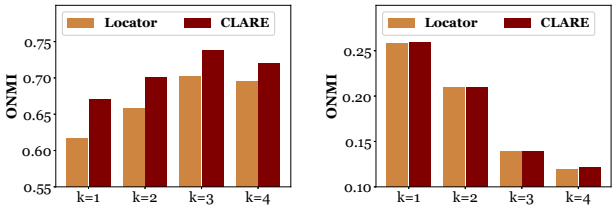
In this section, we emphasize some details of CLARE implementation. Hyper-parameters are summarized in Table 7.

Community Size: From Table 2 we can see some networks are rather dense and sometimes the size of 1-ego net may exceed the maximum size of communities. Therefore, we set the maximum size for generated communities as the maximum size of training ones.

Outer Boundary Size: As mentioned before, some networks are quite dense, the size of a specific outer boundary may by huge, resulting in slow convergence for the optimization of the rewriter. Thus, we fix the maximum size of outer boundary as 10.

Table 7: Hyper-parameters in CLAER

Component	Hyper-parameter	Value
Locator	Batch size	32
	Number of samples in one batch	50
	Number of epochs	2
	Embedding dimension	64
	k & GNN layers	Searched from {1, 2}
	Learning rate	1e-4
	Optimizer	Adam
	Dropout rate	0.2
	Margin α	0.4
Rewriter	MLP of Exclude-Net	65-32-1
	MLP of Expand-Net	65-32-1
	Embedding updater	65-64 GIN
	Optimizer	Adam
	Maximum size of outer boundary	10
	Epoch	1200
	Episode for one epoch	20



(a) Performance on Amazon (b) Performance on DBLP

Figure 8: Comparison with different choices of k

[2]https://pytorch.org/
[3]https://pytorch-geometric.readthedocs.io/
[4]https://snap.stanford.edu/deepsnap/

Table 6: Comparison with different graph neural network encoders. Locator results are reported.

	Amazon			DBLP			Livejournal		
	F1	Jaccard	ONMI	F1	Jaccard	ONMI	F1	Jaccard	ONMI
GCN	**0.7438**	**0.6473**	**0.6586**	0.3819	**0.3116**	**0.2585**	0.4899	0.3953	0.3592
GIN	0.7169	0.6196	0.6313	**0.3841**	0.3100	0.2561	**0.4943**	**0.4004**	**0.3660**
GAT	0.7231	0.6235	0.6318	0.3751	0.3021	0.2446	0.4745	0.3806	0.3405

Algorithm 1: Community Locator Optimization

Input: A graph $G(\mathcal{V}, \mathcal{E}, \mathbf{X})$, the set of training communities $\dot{\mathcal{C}}$

Compute augmented feature matrix $\mathbf{X}'$ based on $\mathbf{X}$
Initialize Encoder Z
while *not converge* **do**
  Generate positive pairs $\mathcal{B}_{pos} = \{(C^i, C^j) | C^i \subset C^j \subseteq \dot{C}^k \in \dot{\mathcal{C}}\}$
  Generate negative pairs $\mathcal{B}_{neg} = \{(C^i, C^j) | C^i \nsubseteq C^j, C^i \subseteq \dot{C}^p, C^j \subseteq \dot{C}^q, \dot{C}^p, \dot{C}^q \in \dot{\mathcal{C}}\}$
  for each pair $(C^i, C^j) \in \mathcal{B}_{pos} \cup \mathcal{B}_{neg}$ **do**
    Encode C^i, C^j as $\mathbf{z}(C^i) = Z(C^i)$, $\mathbf{z}(C^j) = Z(C^j)$
  Traverse over $\mathcal{B}_{pos} \cup \mathcal{B}_{neg}$ to compute $loss$ as Equation 1
  Update Z's parameters using gradient descent to minimize $loss$

Output: Encoder Z

Algorithm 2: Community Rewriter Optimization

Input: Graph $G(\mathcal{V}, \mathcal{E}, \mathbf{X})$, encoder Z, and training communities $\dot{\mathcal{C}}$

Initialize Agent A with parameters ϕ, ψ, θ
while *not converge* **do**
  Generate a set of training samples $\mathcal{D}$
  for each sample $(C^u \cup \partial C^u, \dot{C}^i)$ in $\mathcal{D}$ **do**
    Feed forward $C^u \cup \partial C^u$ to A and obtain the trajectory τ
    Update ϕ, ψ, θ with τ based on Equation 3

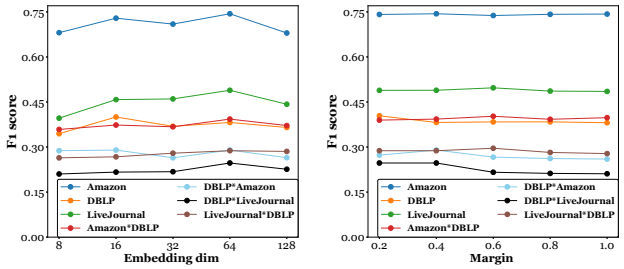
Output: Agent A

Algorithm 3: CLARE Algorithm

Input: A graph $G(\mathcal{V}, \mathcal{E}, \mathbf{X})$, the set of training communities $\dot{\mathcal{C}}(|\dot{\mathcal{C}}| = m)$, and the number of output communities N

Train Encoder Z according to Algorithm 1
$C^{\text{tmp}}, \hat{\mathcal{C}} \leftarrow \emptyset, \emptyset$
$n \leftarrow \frac{N}{m}$
Encode training communities as $\dot{\mathbf{Z}} = \{\mathbf{z}(\dot{C}^i) | \dot{C}^i \in \dot{\mathcal{C}}, i = 1, ..., m\}$
Encode candidate communities as $\mathbf{Z} = \{\mathbf{z}(C^u) | u \in \mathcal{V}\}$
for each $\mathbf{z}(\dot{C}^i) \in \dot{\mathbf{Z}}$ **do**
  Find the set of n closest candidate communities in the embedding space C, $C^{\text{tmp}} = C^{\text{tmp}} \cup C$
Train Agent A according to Algorithm 2
for each $C \in C^{tmp}$ **do**
  Feed C to Agent A and obtain a refined community $\hat{C}$
  $\hat{\mathcal{C}} = \hat{\mathcal{C}} \cup \{\hat{C}\}$

Output: The set of final predicted communities $\hat{\mathcal{C}}$



(a) Different embedding dimensions (b) Different settings of margin α

Figure 9: Parameters study with Locator results reported

B EXPERIMENTS

B.1 Design of Architectures

Design of Encoder in Community Locator: For better architecture and performance, we conduct experiments about the design of encoder in the community locator. We choose three different graph neural networks as the encoder to learn node and community representations, including graph convolutional network (GCN) [19], graph isomorphism network (GIN) [34], and graph attention network (GAT) [12]. For the fairness of comparison, on DBLP, we fix the number of GNN layers as **1** while **2** on Amazon and Livejournal. We report Locator results in Table 6. As can be observed, GCN achieves the best performance on Amazon and DBLP. Although GIN can be competitive on Livejournal, because GIN demands more training time and memory, we choose **GCN** as our encoder finally. Since our encoding objects are small subgraphs, simple graph neural networks are expressive enough [15, 28].

B.2 Parameters Study

In this part, we examine the influence of three key parameters.

Choices of k: Due to communities in different datasets exhibiting distinct features [35], actually k is an important parameter that needs to be tuned. The comparison with different choices of k is depicted in Figure 8. The trend on different datasets varies, as Amazon undergoes performance improvement with the increase of k while DBLP deteriorates. Therefore, we set k=1 for DBLP while k=2 for the remaining datasets during experiments.

Embedding dimension: Figure 9(a) shows the dimension sensitivity experiment results on all datasets. Our model exhibits robustness under different settings of embedding dimensions. In general, CLARE can reach the peak of F1 score with embedding dimension as 64 on most datasets. Therefore, our model chooses 64 as a standard setting.

Margin α: As shown in Figure 9(b), our model is quite stable under different settings of α on all datasets. During experiments, we choose 0.4 as a standard setting for all datasets.