2023 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

PolicyClusterGCN: Identifying Efficient Clusters for Training Graph Convolutional Networks

Saket Gurukar*[1] , Shaileshh Bojja Venkatakrishnan[1], Balaraman Ravindran[2,3] , Srinivasan Parthasarathy[1]
[1] *Department of Computer Science and Engineering*, *The Ohio State University*, Columbus, Ohio
[2] *Indian Institute of Technology Madras*, Chennai, India
[3] *Robert Bosch Centre for Data Science and AI*, Chennai, India

***Abstract*—Graph convolutional networks (GCNs) have achieved huge success in several machine learning (ML) tasks on graph-structured data. Recently, several sampling techniques have been proposed for the efficient training of GCNs and to improve the performance of GCNs on ML tasks. Specifically, the subgraph-based sampling approaches such as ClusterGCN and GraphSAINT have achieved state-of-the-art performance on the node classification tasks. These subgraph-based sampling approaches rely on *heuristics* – such as graph partitioning via edge cuts – to identify clusters that are then treated as minibatches during GCN training. In this work, we hypothesize that rather than relying on such heuristics, one can learn a reinforcement learning (RL) policy to compute efficient clusters that lead to effective GCN performance. To that end, we propose PolicyClusterGCN, an online RL framework that can identify good clusters for GCN training. We develop a novel Markov Decision Process (MDP) formulation that allows the policy network to predict "importance" weights on the edges which are then utilized by a clustering algorithm (Graclus) to compute the clusters. We train the policy network using a standard policy gradient algorithm where the rewards are computed from the classification accuracies while training GCN using clusters given by the policy. Experiments on six real-world datasets and several synthetic datasets show that PolicyClusterGCN outperforms existing state-of-the-art models on node classification task.**

I. Introduction

Graph convolution networks (GCNs) learn high-quality node representations of graph-structured data. Such representations allow GCN to achieve state-of-the-art performance on several graph-based machine learning tasks such as node classification [1], link prediction [2], and recommendation systems [3]. However, GCN cannot easily operate on large-scale graphs as it requires $O(nfl)$ memory [4] where n, f, and l are the number of nodes, features, and GCN layers, respectively.

To improve the efficiency and scaling of GCNs, several sampling-based models have been proposed recently [5]. Liu et al. categorize these sampling methods as node-wise, layer-wise, and subgraph-based sampling models. The recent subgraph-based sampling models such as ClusterGCN [4] and GraphSAINT [6] have shown superior performance on node classification tasks as compared to GCNs and other sampling models. These subgraph-based models identify subgraphs (clusters) that are treated as minibatches for GCN training. The resultant mini-batch training helps improve the performance (such as node classification) of the GCN and also helps in scaling GCNs to large graphs.

Subgraph-based based sampling models rely on *predetermined heuristics* – such as graph partitioning via edge cuts, or communication volume minimization – to identify subgraphs (clusters). For instance, ClusterGCN [4] and DistDGL [7] relies on the clustering algorithm, Metis [8] to identify clusters while Aligraph [9] relies on four graph partitioning algorithms to identify clusters. GraphSAINT [6], on the other hand, relies on random subgraph sampling techniques to identify subgraphs. These techniques achieve better performance than GCN by either avoiding the neighborhood expansion problem [4] in GCN training or by mitigating the high variance in GCN training through normalization [6].

To the best of our knowledge, such heuristics are oblivious to impact of formed clusters on GCN training effectiveness. The optimization objective of GCN [10] suggests that its training performance through clusters is dependent on both the graph structure of the clusters and the distribution of labels within the clusters. However, it is difficult to know *apriori* which heuristics will lead to the identification of efficient clusters for a given input graph. Here, a cluster configuration is efficient if it results in good GCN performance.

Identifying efficient clusters is a hard problem: for k clusters, there exists k^n possible cluster configurations where n is the number of nodes. Training GCN on k^n cluster configurations to find the best performance is not computationally feasible. Hence, we require systematic exploration of the possible configurations. To that end, we propose PolicyClusterGCN, an online reinforcement learning framework that relies on a novel Markov Decision Process (MDP) formulation for computing clusters. The MDP policy is parameterized with a neural network. However, using RL approach to compute efficient clusters is a challenging problem. For instance, a straightforward approach of designing a policy that directly assigns nodes to clusters would lead to clusters with a large fraction of nodes

* This work was done when the author was a student at OSU.

ASONAM '23, November 6-9, 2023, Kusadasi, Turkey

ACM ISBN 979-8-4007-0409-3/23/11…$15.00
http://dx.doi.org/10.1145/3625007.3627499

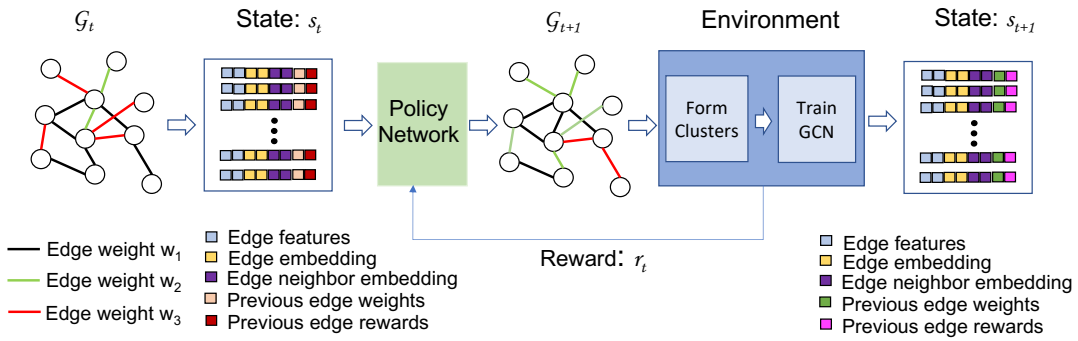


Fig. 1. Overview of PolicyClusterGCN (best viewed in color)

without an immediate edge between them. Hence, we design our policy neural network such that it predicts an "importance" weight for each edge. Edges that are deemed unimportant are assigned a low weight and vice-versa by the policy. A standard edge-cut-based graph partitioning algorithm would then output different cluster configurations based on the edge weights of the graph. This novel setup allows the exploration of a diverse set of clusters while including nodes that are immediately connected in the same clusters. We train the policy network using a standard policy gradient algorithm where the rewards are computed from the classification accuracies while training GCN using clusters given by the policy. Our contributions are summarized as follows:

1) We discover that the choice of clusters has a significant impact on GCN performance.
2) To compute efficient clusters for GCN training, we formulate a novel MDP in which policy predicts edge weights that are utilized by a clustering algorithm to compute clusters.
3) Our experiments show that GCN trained on clusters identified by PolicyClusterGCN outperforms existing methods on five real-world datasets.
4) We also analyze the clusters computed by PolicyClusterGCN by studying their graph structure (via synthetic datasets) and label distribution (via label entropy metric [4]).

II. Notations

Notations: Let $G(V, E)$ be the input graph G with $|V|$ and $|E|$ number of nodes and edges, respectively. Each edge $e = (u, v, w)$ consists of nodes u and v and has edge weight $w \in \mathbb{Z}^+$ that denotes the strength of connection. Let $A \in \mathbb{R}^{|V|\times|V|}$ and $\hat{A} \in \mathbb{R}^{|V|\times|V|}$ be the adjacency matrix and normalized adjacency matrix of graph G, respectively. Let $F \in \mathbb{R}^{|V|\times f}$ be the node features. Let $Z^{(l)}$ be the node embedding at layer l with $Z^0 = F$ and $W^{(l)}$ be the learning parameters of GCN. The node labels are denoted by y_L. Let $\mathcal{T}_\mathcal{G}$ be the training graph consisting of all the training nodes and edges incident on those training nodes. Similarly, let $\mathcal{V}_\mathcal{G}$ and $\mathcal{T}e_\mathcal{G}$ be the validation and test graphs.

III. Methodology

A. MDP formulation

The overview of PolicyClusterGCN is shown in Figure 1. At a step t in MDP, the agent selects an action that updates all the edge weights of the graph in state s_t. As a result, the MDP transitions to new state s_{t+1}. Our MDP is nonepisodic that continuously trains during each step. We experimented with episodic MDP formulations, however, we empirically found our presented MDP design to work better.

Agent: Let $\mathcal{S}$ and $\mathcal{A}$ be the state space and action space, respectively. In this work, a state represents the graph. Let α be the set of possible discrete edge weights on an edge and $a = (e_1, \ldots, e_{|E|})$ be the selected edge weights where $e_i \in \alpha$. The edge placement policy can then be defined as a mapping $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that assigns an edge weight to all the edges where $\mathcal{A} = \alpha^{|E|}$. The goal of the policy network is to find a placement policy π that results in efficient clusters.

Environment: The environment accepts a graph and its edge weights computed by the agent as its input. It then partitions the graph into several clusters based on the edge weights and trains the GCN model on each cluster.

State: A state observation $s \in S$ comprises of training graph $\mathcal{T}_\mathcal{G}$ with the following features on each edge $e = (u, v)$: (1) concatenation of node features of nodes u and v; (2) concatenation of node embeddings of nodes u and v. Here, we utilize an unsupervised graph representation learning method (UGRL), node2vec [11] to learn the node embeddings (later, we show in the experiments section, that one can also choose other UGRL methods for this step); (3) concatenation of embeddings of 1-hop neighbor nodes of nodes u and v. A node's neighbor embeddings are aggregated with sum operation. We choose the sum operation as it allows us to capture the graph neighborhood in an expressive manner [12]; (4) the previous m number of edge weights taken by the policy π; (5) the previous m number of rewards received from the environment. At the initial state s_0, the previous m edge weights and previous m edge rewards are assigned a value of 1 and 0, respectively.

Action: An action at step t is given by the policy π: $a_t = \pi(s_t) = (e_1, \ldots, e_{|E|})$. To accelerate the exploration phase through actions, we utilize exponential edge weights. In

Algorithm 1 PolicyClusterGCN

Input: $\mathcal{T}_\mathcal{G}$: train graph, $\mathcal{V}_\mathcal{G}$: validation graph, k : num clusters, iters: clustergcn iterations, α : set of discrete edge weights, y_{T_L} and y_{V_L}: train and validation node labels, respectively.
Parameters: γ, θ, w, α^θ, α^w, ϵ, $\epsilon_{start}, \epsilon_{end}, \epsilon_{decay}$
Output: Cluster configuration.

for t = 0, 1, 2, ... T **do**
  Assign edge weight of edge i where $1 \leq i \leq |E|$
  $$w_i = \begin{cases} \pi_\theta(s_t)[i] & \text{with } 1-\epsilon \text{ prob.} \\ random(\alpha) & \text{with } \epsilon \text{ prob.} \end{cases}$$
  Decay ϵ at each step
  Update graph ($\mathcal{T}_{\mathcal{G}'}$) with new edge weights.
  Identify k clusters from ($\mathcal{T}_{\mathcal{G}'}$). Change edge weight of k clusters to that of $\mathcal{T}_\mathcal{G}$
  Train ClusterGCN in Algorithm 2 on the identified clusters of $\mathcal{T}_\mathcal{G}$.
  Compute validation score from the above trained ClusterGCN on $\mathcal{V}_\mathcal{G}$.
  if validation-score > best-validation-score: **then**
    cluster_configuration = identified clusters
    best-validation-score = validation-score
  end if
  Compute score of edge i = (u,v) as $sc_i = sc_u + sc_v$
  $$sc_u = \begin{cases} +1 & \text{if node label predicted correctly} \\ -1 & \text{otherwise} \end{cases}$$
  $r_t = \frac{1}{|E|} \sum sc_i$ where $1 \leq i \leq |E|$
  Update parameters θ and w using equations 1 and 2.
end for
return cluster_configuration

exponential edge weights, if there are p number of possible actions, an action i would correspond to edge weight 2^i where $0 \leq i \leq p$. An edge with 2^p weight would have the strongest connection and would be less likely to be eliminated during graph partitioning. The choice of discrete edge weights instead of continuous edge weights for the actions is primarily due to the restriction-induced by graph partitioning algorithms (e.g. Metis [8], Graclus [13], and MLR-MCL [14]).

Reward: Now, to identify a good cluster configuration from the space of all possible cluster configurations, we require a reward signal about the goodness of a given cluster configuration. Here, we treat the performance (such as node classification) of GCN on training nodes as the reward signal by training GCN for a few iterations ($iters$) over the computed clusters. We train GCN for a small number of iterations (for example, 50 iterations in our experiments) to get rewards in a short amount of time. Once the GCN model is trained, we compute a score for each edge $e = (u, v)$ of the training graph by summing scores of nodes u and v. Here, a score for a node is +1 if the node label is predicted correctly by the trained GCN while -1 otherwise. In the case of multilabel classification, we compute the score for each label and then sum the scores. We compute the reward as the mean of all the scores across all the edges.

Algorithm 2 ClusterGCN

Input: $\mathcal{T}_\mathcal{G}{}_1, \mathcal{T}_\mathcal{G}{}_2, ...\mathcal{T}_\mathcal{G}{}_k$; iters, l

Let $\mathcal{B} = \{\mathcal{T}_\mathcal{G}{}_1, \mathcal{T}_\mathcal{G}{}_2, ...\mathcal{T}_\mathcal{G}{}_k\}$
for iter = 1, ... $\leq$ iters **do**
  $\mathcal{B} \leftarrow$ Permute $\mathcal{B}$ in an uniform random manner.
  for x in $\mathcal{B}$ **do**
    $Z_x^{(l+1)} = \sigma(\hat{A}_{\mathcal{G}x} Z_x^{(l)} W^{(l)})$ where $Z_x^{(0)} = F_x$
    minimize $\mathcal{L} = \frac{1}{|y_L|} \sum_{i \in y_L} loss(y_i, Z_{x,i}^L)$
  end for
end for

B. Policy Network Architecture

PolicyClusterGCN learns the good cluster configuration of a graph by parameterizing the MDP policy using a neural network. Here, one could design the policy network using GCN. However, GCN has a huge memory requirement for large graphs. Hence, we select a two-layer neural network as our policy network. The input to the network is the state of edge i and the output of the network is the predicted edge weight. This design allows the parallelization of the edge weight prediction task. This policy network design can also perform the edge weight prediction task for large graphs.

The policy network is trained with a standard policy gradient algorithm.

C. Training

PolicyClusterGCN is trained with a standard policy gradient algorithm - actor-critic [15]. Let θ and w be the policy parameters (actor) and state-value function parameters (critic), respectively. Let r_t be the received reward obtained by following policy π_θ at step t. Then the actor and critic parameters are updated as follows:

$$\delta \leftarrow r_t + \gamma \hat{v}(s_{t+1}, w) - \hat{v}(s_t, w), \tag{1}$$

where γ is the discount factor and $\hat{v}$ is the value function and,

$$\begin{aligned} w &\leftarrow w + \alpha^w \delta \ \nabla_w \ \hat{v}(s_t, w) \\ \theta &\leftarrow \theta + \alpha^\theta \delta \ \nabla_\theta \ \ln \pi(a_t | s_t, \theta), \end{aligned} \tag{2}$$

where $\alpha^w > 0$ and $\alpha^\theta > 0$ are the step sizes for actor and critic parameters and $\mathcal{A}_t$ denotes the actions taken by step t.

The training algorithm of PolicyClusterGCN is presented in Algorithm 1. In each step, we update the edge weights of all the edges. Here, given the state s_t at step t, we employ an ϵ-greedy method for exploring the edge weights. We set the edge weight of an edge i to one of the possible edge weights in a uniform random manner with ϵ probability and set the edge-weight of the edge given by the policy network

$(\pi_\theta(s_t)[i])$ with $1-\epsilon$ probability. We decay the ϵ values as $\epsilon_{end} + (\epsilon_{start} - \epsilon_{end}) \times \exp(-1 \times t/\epsilon_{decay}))$. We update the training graph $\mathcal{T}_\mathcal{G}$ with the predicted edge weights.

The change in the edge weights of the graph allows us to explore different cluster configurations. We identify clusters using a multi-level graph partitioning algorithm, Graclus [13] that allows the identification of imbalanced clusters. Metis [8], on the other hand, has a load balance constraint that allows limited exploration of diverse cluster configurations. Once, we identify the clusters, we change the edge weight of the non-cut edges to that of the original graph. This change is important as we are interested in identifying good cluster configuration rather than modifying the input graph.

Next, to compute the reward signal we train the ClusterGCN model (shared in Algorithm 2). In ClusterGCN, given k clusters, we form $\mathcal{T}_{\mathcal{G}1}, \mathcal{T}_{\mathcal{G}2}, ...\mathcal{T}_{\mathcal{G}k}$ subgraphs. Let $\hat{A}_{\mathcal{G}x} \in \mathbb{R}^{n_x \times n_x}$ be the normalized adjacency matrix of subgraph x with n_x number of nodes. Let $Z_x^{(l)} \in \mathbb{R}^{n_x \times f}$ be the node embedding at layer l with $Z_x^0 = F_x$ where F_x is the initial node features. While $W^{(l)}$ are the learnable parameters. Then, we train GCN by following the training procedure outlined in ClusterGCN [4]. The ClusterGCN training process also includes a parameter ($bsize$) to enable a stochastic multi-partition approach and we follow the same approach in our implementation (approach [4], not shown in Algorithm 2 for expository simplicity). Once we train ClusterGCN, we compute the reward by following the procedure mentioned in "MDP formulation" section. We also compute the validation score on the supervised task (such as node classification) and store the clusters that resulted in the best validation score. The policy and critic networks are then trained using equations 1 and 2.

IV. Experiments

A. Datasets

We evaluate PolicyClusterGCN on six frequently evaluated datasets. The statistics of these datasets are presented in Table I. The column train/val/test in Table I refers to the size of training, validation, and test splits of the dataset. The Romania, Hungary, and Croatia datasets were introduced in [16] and extensively used in the literature. We evaluate models on multilabel classification tasks on Romania, Hungary, Croatia, Twitter, and Facebook datasets and multiclass classification on the Blogcatalog dataset. In the case of Twitter and Facebook datasets, certain labels are present in only a few nodes, hence in our evaluation, we consider only those labels that are present in at least 10 nodes. Node attributes are not present for Romania, Hungary, and Croatia datasets. Hence, we factorize the adjacency matrix of these datasets and treat the 16 left singular vectors as node attributes [17].

TABLE I
Dataset statistics. Feats denotes number of features. L/C denotes number of labels or classes.

Datasets	Nodes	Edges	Feats	L/C	train/ val/ test	References
Romania	41,773	996,404	16	84	0.60 / 0.10 / 0.30	[16],
Hungary	47,538	445,774	16	84	0.60 / 0.10 / 0.30	[24]–
Croatia	54,573	251,652	16	84	0.60 / 0.10 / 0.30	[30]
Twitter	2,403	37,154	9,073	73	0.55 / 0.20 / 0.25	[31]–[33]
Facebook	4,039	88,234	1,283	84	0.60 / 0.20 / 0.20	[31], [34], [35]
Blogcatalog	5,196	343,486	8189	6	0.55 / 0.20 / 0.25	[36]–[39]

B. Baselines

We consider state-of-the-art GCN training models including several models that perform efficient sampling for GCN training [5]. These baselines include: **GCN** [10], **GraphSAGE** [18], **VR-GCN** [19], **FAST-GCN** [20], **LADIES** [21], **RippleWalk** [22], **GraphSaint** [6], **ClusterGCN** [4]. We compare performance of these baselines against **PolicyClusterGCN**.

We observe that GraphSaint utilizes a higher-order GCN (HOGCN) module [23] instead of GCN module. Hence, for fair comparison, we also perform experiments with **ClusterHOGCN** and **PolicyClusterHOGCN** where we replace GCN module in these methods with HOGCN.

C. Training Details

PolicyClusterGCN is implemented in PyTorch. For baselines we utilize the code provided by the authors. For PolicyClusterGCN we apply Graclus [13] partitioning method to formulate clusters.

We set the number of GCN layers to 2 for all the baselines. We tune the following hyper-parameters for all the baselines: learning rates = [0.01, 0.001], dropouts = [0.0, 0.1, 0.2] and embedding dimensions = [128, 512, 1024]. The number of epochs is set to 1500 for all the models. For GraphSAINT, we also tune the normalization parameters = ["norm", "norm-nn"] and embedding aggregation process = ["mean", "concatenation"]. For ClusterGCN and PolicyClusterGCN, we tune the number of clusters = [4, 8, 16, 32] and number of clusters for each batch = [1, 4, 8]. For all the models, we perform the evaluation on the same five train/val/test splits and report the average test micro-f1 score over five runs. For PolicyClusterGCN, we set the number of previous edge weights and the number of the previous edge rewards value m to 5. To get the node embeddings, we set the node2vec parameters as walk-length=40, context-size=5, walks-per-node=20, p=1.0, q=1.0. During development, we played with the parameters of node2vec and found that PolicyClusterGCN is not sensitive to those parameters. We train PolicyClusterGCN on the cluster configuration identified by Algorithm 1.

PolicyClusterGCN parameters: We set the eps_decay value to 100. The step sizes of $\alpha^w = 0.001$ and $\alpha^\theta = 0.001$. The discount factor γ is set to 0.95. During development, we played with the following node2vec parameters: walk-length=[20, 40, 80], context-sizes=[3,5], walks-per-node=[20, 40]. However, we did not find PolicyClusterGCN to be sensitive with respect to node2vec parameters.

All the experiments are conducted on a machine with an NVIDIA Tesla V100 (32 GB memory), an Intel Xeon E5-2680 CPU (28 cores, 2.40GHz), and 128 GB of RAM.

TABLE II
TEST MICRO-F1 SCORE ON THE NODE CLASSIFICATION TASK. RESULTS AVERAGED OVER FIVE INDEPENDENT RUNS FOR ALL THE MODELS.

Sampling Strategy	Model	Croatia	Romania	Hungary	Facebook	Twitter	Blogcatalog
	GCN	0.343 (± 0.030)	0.340 (± 0.034)	0.384 (± 0.036)	0.500 (± 0.031)	0.150 (± 0.015)	0.940 (± 0.012)
Node wise sampling	GraphSage	0.355 (±0.011)	0.352 (±0.017)	0.399 (±0.024)	0.478 (±0.022)	0.132 (±0.022)	0.933 (±0.005)
	VR-GCN	0.338 (± 0.026)	0.366 (± 0.030)	0.396 (± 0.030)	0.525 (± 0.010)	0.156 (± 0.055)	0.953 (± 0.010)
Layer wise sampling	LADIES	0.403 (± 0.040)	0.377 (± 0.043)	0.422 (± 0.043)	0.459 (±0.024)	0.137 (±0.023)	0.889 (±0.006)
	FAST-GCN	0.404 (± 0.075)	0.375 (± 0.070)	0.383 (± 0.056)	0.368 (±0.009)	0.115 (±0.034)	0.832 (±0.032)
Subgraph sampling	RippleWalk	0.359 (± 0.027)	0.358 (± 0.030)	0.406 (± 0.0340)	0.466 (± 0.046)	0.144 (± 0.052)	0.877 (± 0.011)
	GraphSaint	0.459 (±0.005)	0.467 (±0.002)	0.485 (±0.001)	0.537 (±0.007)	0.164 (±0.014)	**0.959** (±0.005)
	ClusterGCN	0.403 (±0.012)	0.387 (±0.005)	0.420 (±0.043)	0.510 (±0.006)	0.148 (±0.032)	0.950 (±0.004)
	ClusterHOGCN	0.453 (±0.003)	0.469 (±0.006)	0.488 (±0.001)	0.557 (±0.013)	0.171 (±0.022)	0.951 (±0.004)
Proposed	PolicyClusterGCN	0.428 (±0.009)	0.413 (±0.001)	0.453 (±0.007)	0.515 (±0.008)	0.155 (±0.024)	0.952 (±0.006)
	PolicyClusterHOGCN	**0.463** (±0.003)	**0.478** (±0.003)	**0.497** (±0.001)	**0.563** (±0.001)	**0.189** (±0.016)	0.956 (±0.006)

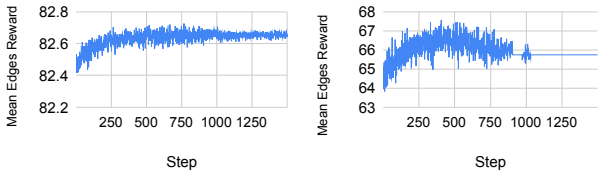
V. RESULTS

A. Performance

We compare the node classification performance of PolicyClusterGCN with other baselines in Table II. The values in the parenthesis in Table II represents standard deviation. We rely on the Micro-f1 metric for evaluation as it is a frequently used metric in the literature. We observe that our proposed PolicyClusterHOGCN outperforms state-of-the-art models on five out of six real-world datasets. Moreover, our proposed PolicyClusterHOGCN and PolicyClusterGCN outperform ClusterHOGCN and ClusterGCN on all the datasets. This result suggests that our approach is able to identify strong cluster configurations than prior state-of-the-art approaches. The difference in performance between PolicyClusterHOGCN and other baselines is statistically significant with a standard paired t-test at a significance level of 0.05. Another observation is that with HOGCN, the performance of ClusterHOGCN is close to or often better than that of GraphSAINT. Note that our proposed framework is not limited to GCN or HOGCN and one can experiment with other graph convolutional networks design [40] for further improvement in the supervised task.

B. Reward Analysis

Figure 2 shows the mean of mean edge rewards received over five independent runs. We observe that as the policy network's training progresses, PolicyClusterGCN can identify better cluster configurations that result in the improvement in GCN training performance.



(a) Facebook (b) Twitter

Fig. 2. Mean edges reward per PolicyClusterHOGCN step

C. Robustness: State Embedding Methods

We test the sensitivity of the proposed PolicyClusterGCN framework with respect to the choice of state embedding methods. We consider three UGRL methods: a random-walk based method, node2vec [11], a community aware embedding method, M-NMF [41], and a matrix factorization based approach, NetMF [42]. The average results over five independent runs are presented in Table III. We observe that PolicyClusterHOGCN achieves similar performance on all of the six real-world datasets and is robust to the choice of state embedding method.

TABLE III
ROBUSTNESS: STATE EMBEDDING METHODS.

		Croatia	Romania	Hungary	Facebook	Twitter	Blogcatalog
PolicyClusterHOGCN	Node2vec	0.463	0.478	0.497	0.563	0.189	0.956
	M-NMF	0.461	0.476	0.497	0.556	0.163	0.956
	Net-MF	0.462	0.478	0.498	0.562	0.168	0.956

TABLE IV
ROBUSTNESS: POLICY GRADIENT METHODS.

		Croatia	Romania	Hungary	Facebook	Twitter	Blogcatalog
PolicyClusterHOGCN	Actor-critic	0.463	0.478	0.497	0.563	0.189	0.956
	Reinforce	0.461	0.479	0.488	0.562	0.177	0.955

TABLE V
SYNTHETIC DATASET: LFR. LFR_μ CORRESPONDS TO PERCENTAGE OF INTER CLUSTER LINKS.

LFR_μ	ClusterGCN	PolicyClusterGCN	% diff.
0.10	0.711	0.731	+2.00
0.15	0.739	0.753	+1.40
0.20	0.552	0.561	+0.90
0.25	0.452	0.472	+2.05
0.30	0.223	0.237	+1.40
0.35	0.193	0.201	+0.80
0.40	0.135	0.144	+0.90

D. Robustness: Policy Gradient Methods

Next, we test the sensitivity of our framework with respect to the training algorithm. Here, we train PolicyClusterHOGCN with reinforce [43] and actor-critic [15] algorithms. Table IV shows the performance of PolicyClusterHOGCN. We observe that PolicyClusterHOGCN trained with actor-critic often outperforms PolicyClusterHOGCN trained with the reinforce algorithm. The performance improvement is likely due to the better ability of the actor-critic algorithm to reduce gradient variance [44].

E. Synthetic Datasets

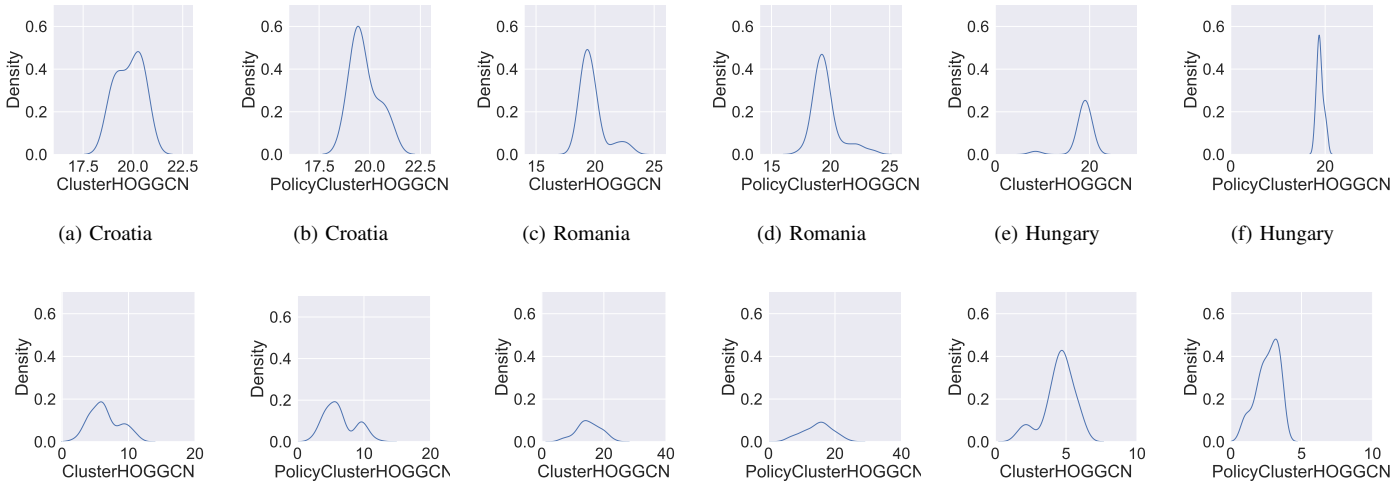
We compare the performance of PolicyClusterHOGCN and ClusterHOGCN on the synthetic LFR datasets [45]. Each LFR dataset consists of a set of communities and μ percentage of inter-community connections. In this section, we study how the performance of PolicyClusterHOGCN and PolicyClusterHOGCN changes as we change the value of μ from 0.1 to 0.4 with a 0.05 step size. We keep the rest of the parameters of the LFR dataset the same (number of nodes=5,000, average degree=5, min community size=50, power law exponent for the degree distribution and community size distribution to 3 and 1.5, respectively). We use cdlib library [46] for generating the datasets. A node's cluster is set as its class and we perform multi-class classification. For node features, we factorize the adjacency matrix of these datasets and treat 16 left singular vectors as node attributes. The test Micro-f1 scores are shared in Table V. We observe that PolicyClusterHOGCN is consistently able to identify better clusters as compared to ClusterHOGCN for all μ values. However, as the percentage of inter-community edges increases, the performance between PolicyClusterHOGCN and ClusterHOGCN decreases. This result is not terribly surprising since at that point noise (represented by inter-cluster edges) tends to dominate the signal (represented by node clusters).

F. Cluster Analysis: Label Entropy

The performance of GCN is dependent on both cluster structure and node label distribution in the cluster. Here, we compare and contrast the node labels present in the clusters identified by ClusterHOGCN and PolicyClusterHOGCN. For comparison, we rely on the label entropy metric utilized elsewhere [4]. To compute a cluster's label entropy, we assume labels are independent and a node's label can have 0 or 1 value. For q labels, cluster c's label entropy can then be calculated as $S_c = H(X_1, X_2, ..., X_q) = H(X_1) + H(X_2) + ... + H(X_q)$ where X_i is Bernoulli random variable. A dataset's label entropy is represented through the distribution of cluster label entropies and consists of k number of clusters. Since we report the results over five independent runs, our reported distribution size is k times five. Figure 3 presents the distribution of cluster label entropies using the kernel density estimation algorithm for six real-world datasets. We observe that, in general, the distribution of label entropies of ClusterHOGCN has high variance as compared to that of PoliyClusterHOGCN which has low variance – see Figure 3a vs 3b, Figure 3e vs 3f, Figure 3k vs 3l. The low variance or high saturated label entropy's suggests that PolicyClusterGCN identifies clusters with high label uncertainty in the clusters.

VI. RELATED WORK

Graph convolutional neural networks (GCNs) have shown promising results on several machine learning tasks on graph-structured data. However, GCNs require the full normalized



(a) Croatia (b) Croatia (c) Romania (d) Romania (e) Hungary (f) Hungary

(g) Facebook (h) Facebook (i) Twitter (j) Twitter (k) Blogcatalog (l) Blogcatalog

Fig. 3. Label Entropy

adjacency matrix of the graph, and hence scaling them on large networks becomes challenging [4], [7], [18].

A plethora of efficient sampling methods has been proposed to solve the scalability bottleneck and improve the GCN training [5]. The sampling-based techniques can be categorized as node sampling vs layer sampling vs subgraph sampling. In node sampling techniques, one usually samples a fixed number of neighbors per node and forms a subgraph consisting of multiple connected nodes and their sampled neighbors [18]. VR-GCN utilizes the historical activations of nodes in GCN training to efficiently sample a much smaller number of neighbors per node. Layer sampling-based method sample usually diverse nodes per GCN layer [20]. LADIES [21] improved the sampling process through importance sampling.

Subgraph sampling-based GCN approaches have achieved state-of-the-art performances on the node classification task. These approaches often perform clustering using graph partitioning algorithms [4], [7], [9] and the identified subgraph is treated as a minibatch for GCN training. RippleWalk [22] proposed a novel set expansion-based subgraph sampling method to construct minibatches. GraphSAINT [6] proposed several subgraph samplers to identify subgraphs and also introduced several normalization techniques to reduce the bias and variance of GCN training.

While we have not evaluated its use for the placement of operations on computational devices we believe PolicyClusterGCN can be used for such a purpose (see Placeto [47]). Placeto [47] learns efficient placement of node (tensorflow operations) on a compute device (GPUs). Placeto's single episode has an episode length equal to the number of nodes in the graph. One episode step places a node to a different cluster and then passes the modified graph to the environment to get the reward – rendering the training process to be quite expensive. We believe PolicyClusterGCN can simplify this task and plan to examine such ideas in the future.

Conclusion

We propose PolicyClusterGCN, an online RL based approach, to identify such efficient cluster configuration for GCN training. PolicyClusterGCN's policy network modifies the edge weights of the graph that allows it to explore diverse cluster configurations. We train the policy network using an actor-critic algorithm where the reward signal is received through GCN performance. We perform experiments on six real-world datasets and show that our proposed model can outperform state-of-the-art baselines.

In the future, we plan to explore two directions for PolicyClusterGCN: add generalizability functionality and improve scalability. To add the generalizability functionality, we propose to transform the nodes embedding of different graphs into a common embedding space and devise an efficient training algorithm for PolicyClusterGCN. To improve scalability, we plan to explore multi-level frameworks that rely on graph coarsening to significantly reduce the size of the graph [48], [49].

VII. Acknowledgements

SG and SP were partially supported by the National Science Foundation (NSF) under grants CNS-2112471 (which includes an international collaboration supplement with BR), OAC-2018627, and CCF-2028944. Any opinions, findings, and conclusions in this material are those of the author(s) and may not refect the views of the respective funding agencies.

References

[1] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014.

[2] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[3] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *KDD*, 2018, pp. 974–983.

[4] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *KDD*, 2019, pp. 257–266.

[5] X. Liu, M. Yan, L. Deng, G. Li, X. Ye, and D. Fan, "Sampling methods for efficient training of graph convolutional networks: A survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 2, pp. 205–234, 2021.

[6] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," *arXiv preprint arXiv:1907.04931*, 2019.

[7] D. Zheng, C. Ma, M. Wang, J. Zhou, Q. Su, X. Song, Q. Gan, Z. Zhang, and G. Karypis, "Distdgl: distributed graph neural network training for billion-scale graphs," in *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*. IEEE, 2020, pp. 36–44.

[8] G. Karypis and V. Kumar, "Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," 1997.

[9] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "Aligraph: a comprehensive graph neural network platform," *arXiv preprint arXiv:1902.08730*, 2019.

[10] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[11] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[12] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[13] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007.

[14] V. Satuluri and S. Parthasarathy, "Scalable graph clustering using stochastic flows: applications to community discovery," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 737–746.

[15] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[16] B. Rozemberczki, R. Davies, R. Sarkar, and C. Sutton, "Gemsec: Graph embedding with self clustering," in *Proceedings of the 2019 IEEE/ACM international conference on advances in social networks analysis and mining*, 2019, pp. 65–72.

[17] W. Cukierski *et al.*, "Graph-based features for supervised link prediction," in *The 2011 International joint conference on neural networks*. IEEE, 2011, pp. 1237–1244.

[18] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[19] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *International Conference on Machine Learning*, 2018, pp. 941–949.

[20] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *arXiv preprint arXiv:1801.10247*, 2018.

[21] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," *Advances in neural information processing systems*, vol. 32, 2019.

[22] J. Bai, Y. Ren, and J. Zhang, "Ripple walk training: A subgraph-based training framework for large and deep graph neural network," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[23] S. Abu-El-Haija and et al., "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *ICML*. PMLR, 2019, pp. 21–29.

[24] L. Wang, B. Zong, Q. Ma, W. Cheng, J. Ni, W. Yu, Y. Liu, D. Song, H. Chen, and Y. Fu, "Inductive and unsupervised representation learning on graph structured objects," in *International conference on learning representations*, 2019.

[25] B. Rozemberczki, O. Kiss, and R. Sarkar, "Little ball of fur: a python library for graph sampling," in *CIKM*, 2020, pp. 3133–3140.

[26] H. Yang, Q. Kong, W. Mao, and L. Wang, "Boosting hidden graph node classification for large social networks," in *ISI*. IEEE, 2021, pp. 1–6.

[27] G. Bianconi, H. Sun, G. Rapisardi, and A. Arenas, "Message-passing approach to epidemic tracing and mitigation with apps," *Physical Review Research*, vol. 3, no. 1, p. L012014, 2021.

[28] F. Zhou, C. Su, S. Xu, and L. Lv, "Influence fast or later: Two types of influencers in social networks," *Chinese Physics B*, vol. 31, no. 6, p. 068901, 2022.

[29] V. Auletta, D. Ferraioli, V. Fionda, and G. Greco, "Maximizing the spread of an opinion when tertium datur est," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019, pp. 1207–1215.

[30] A. Bouyer and H. A. Beni, "Influence maximization problem by leveraging the local traveling and node labeling method for discovering most influential nodes in social networks," *Physica A: Statistical Mechanics and its Applications*, vol. 592, p. 126841, 2022.

[31] T. He and K. C. Chan, "Discovering fuzzy structural patterns for graph analytics," *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 5, pp. 2785–2796, 2018.

[32] T. He, K. C. Chan, and L. Yang, "Clustering in networks with multimodality attributes," in *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. IEEE, 2018, pp. 401–406.

[33] L. Yao and T. He, "Fuzzy community detection with multi-view correlated topics," in *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2021, pp. 307–313.

[34] L. Hu, X. Pan, and X. Luo, "Incorporating generalized momentum method to accelerate clustering analysis of complex networks," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2021, pp. 2051–2056.

[35] Z. Tang, L. Hu, and X. Pan, "Detection of link communities in attributed graphs via an approximate bayesian generative model," in *2022 5th International Conference on Artificial Intelligence and Big Data (ICAIBD)*. IEEE, 2022, pp. 315–320.

[36] X. Huang, Q. Song, Y. Li, and X. Hu, "Graph recurrent networks with attributed random walks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 732–740.

[37] X. Huang, Q. Song, J. Li, and X. Hu, "Exploring expert cognition for attributed network embedding," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 2018, pp. 270–278.

[38] X. Huang, J. Li, and X. Hu, "Accelerated attributed network embedding," in *Proceedings of the 2017 SIAM international conference on data mining*. SIAM, 2017, pp. 633–641.

[39] X. e. a. Huang, "Label informed attributed network embedding," in *Proceedings of the tenth ACM international conference on web search and data mining*, 2017, pp. 731–739.

[40] J. You, Z. Ying, and J. Leskovec, "Design space for graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 17 009–17 021, 2020.

[41] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *AAAI*, 2017.

[42] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *Proceedings of the eleventh ACM international conference on web search and data mining*, 2018, pp. 459–467.

[43] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[44] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[45] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Physical review E*, vol. 78, no. 4, p. 046110, 2008.

[46] G. Rossetti, L. Milli, and R. Cazabet, "Cdlib: a python library to extract, compare and evaluate communities from complex networks," *Applied Network Science*, vol. 4, no. 1, pp. 1–26, 2019.

[47] R. Addanki, S. B. Venkatakrishnan, S. Gupta, H. Mao, and M. Alizadeh, "Placeto: Learning generalizable device placement algorithms for distributed machine learning," *arXiv preprint arXiv:1906.08879*, 2019.

[48] C. Deng, Z. Zhao, Y. Wang, Z. Zhang, and Z. Feng, "Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding," *arXiv preprint arXiv:1910.02370*, 2019.

[49] J. Liang *et al.*, "Mile: A multi-level framework for scalable graph embedding," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 15, 2021, pp. 361–372.