Community Detection using Semi-supervised Learning with Graph Convolutional Network on GPUs

Naw Safrin Sattar and Shaikh Arifuzzaman
Department of Computer Science
University of New Orleans
New Orleans, LA-70148, USA.
Email: {nsattar, smarifuz}@uno.edu

2020 IEEE International Conference on Big Data (Big Data) | 978-1-7281-6251-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/BigData50022.2020.9378123

***Abstract*—Graph Convolutional Network (GCN) has drawn considerable research attention in recent times. Many different problems from diverse domains can be solved efficiently using GCN. Community detection in graphs is a computationally challenging graph analytic problem. The presence of only a limited amount of labelled data (known communities) motivates us for using a learning approach to community discovery. However, detecting communities in large graphs using semi-supervised learning with GCN is still an open problem due to the scalability and accuracy issues. In this paper, we present a scalable method for detecting communities based on GCN via semi-supervised node classification. W e o ptimize t he h yper-parameters f or our semi-supervised model for detecting communities using PyTorch with CUDA on GPU environment. We apply Mini-batch Gradient Descent for larger datasets to resolve the memory issue. We demonstrate an experimental evaluation on different real-world networks from diverse domains. Our model achieves up to 86.9% accuracy and 0.85 F1 Score on these practical datasets. We also show that using identity matrix as features, based on the graph connectivity, performs better with higher accuracy than that of vertex-based graph features. We accelerate the model performance 4 times with the use of GPUs over CPUs.**

***Index Terms*—graph convolutional network, community detection, graph problems, semi-supervised learning, machine learning, deep learning, neural network, optimization, GPU**

I. Introduction

Community detection in graphs (or graph clustering) is an important graph analysis kernel with many real-world applications in diverse socio-technical domains including sociology, biology, infrastructure, web, and epidemiology [1]–[3]. For example, clustering helps studying the effect of rumor or epidemic spreading in social or population network. Functional units in protein-interaction networks can also be found by community detection [4]. In general, community detection kernel identifies large-scale map of a network where individual communities act like meta-nodes or functional units. Communities reveals significant insights on the organization, function, or structural characterization of a system represented by graph [1].

Nowadays, we experience a huge growth of data generating from data-driven scientific and technical disciplines, e.g., online social media, biological sciences, business systems, and the Internet [5]. Such large datasets (including graph/network data) necessitate scalable methods for efficient and effective sense-making [6], [7].

Due to the advancement of data technology and computing resources, machine learning, particularly deep learning, has become a popular and exciting area of research for a growing number of applications [8]–[11]. Graph Convolutional Network (GCN) combines both graph algorithms and deep learning and provides a wide avenue of practical applicability. GCN has demonstrated great promise in mining graph data (citation, social, and protein-protein interaction networks) [9], predicting protein interface [12], disease classification [13], molecules discovery in chemistry, text classification [14], identifying traffic congestion, image classification [15], [16], etc.

Using semi-supervised learning we can achieve the same or comparable result with less computational cost compared to a traditional community detection method. Community detection has several applications where this computational cost is high because of large datasets. Estimating unknown features of users/entities in social networks is a common application of community detection. If we think of Facebook network, it has over 2.6 billion active users. Traditional community detection algorithms need to analyze the full network to detect the communities and is computationally expensive. In case of semi-supervised learning, the main advantage is that if we have community information of a part of the network, we can predict the rest communities depending on those community labels for such large networks.

In this work, we use graph convolutional network to detect communities in large graphs using semi-supervised node classification. Our key contributions are as follows.

- **Model:** We present a scalable method for detecting communities via semi-supervised node classification. We apply Mini-batch Gradient Descent to solve the memory issues of GCN for larger and denser networks. We use Identity Matrix as Feature set and achieve better performance compared to node-based Graph Feature Set.

- **Data:** We experiment with a diverse set of real-world networks from different domains. Our model achieves around 86.9% accuracy and 0.85 F1 Score. For the popular datasets, our model performs comparably with the state of the art and even better in several cases.
- **Experiment:** We provide an extensive and thorough experimental evaluation of our model. We perform an optimization on the hyper parameters for better accuracy. We demonstrate a comparison of our method with related work empirically. We also accelerate the model performance 4 times using GPUs.

The rest of this paper is organized as follows. We describe the existing related works in Section II. Preliminary introduction on Graph Convolutional Network is presented in Section III. We describe our methods to generate features and train the model in Section IV. A detailed analysis of the results, e.g., description of our datasets, experimental setting, model performance, optimization of the hyper parameters, improvement of model accuracy, model speedup with parallelization, etc., are described in Section V .

II. Related Works

There has been a rich literature of community detection in networks [1]–[3], [7], [17]–[21]. Most of these works are computationally expensive and do not scale well to large-sized networks. Kipf et al. [9] demonstrate some promising results based on GCN for graph clustering, albeit using a small dataset (Zachary's karate club network). Both modularity-based clustering [22] and untrained 3-layer GCN Model [9] detect the same clusters for the said network. The authors [9] did not extend their work for large and more diverse datasets.

Another work by Chen et al. [23] used graph neural network (GNN) to detect communities with supervised learning. The paper restricted the largest community size to 800 nodes. They used networks having average community size below 30 in their experiments. Thus, an extensive experimentation with large-scale graphs with variety of domain and structural properties is still missing in literature.

GraphSAGE [10] is a framework designed to efficiently generate representations on evolving graphs with previously unseen data leveraging node attribute information. The *mean aggregator* used by the paper is nearly equivalent to the convolutional propagation rule used in the GCN framework [9]. In experimentation, they omit the largest communities from the networks. LAyer-Dependent ImportancE Sampling (LADIES) [24] is also a sampling based method. This algorithm selects their neighborhood nodes depending on the sampled nodes in the upper layer and constructs a bipartite subgraph. Then, the method computes the importance probability accordingly and samples a fixed number of nodes based on this probability. But the authors limited their experimentation to a few small-sized dataset. Another framework, Neural Overlapping Community Detection (NOCD) is a graph neural network model for overlapping community detection [25]. They use the Bernoulli–Poisson (BP) model that allows for overlapping communities. One major difference between their model and GCN [9] is that they apply batch normalization after the first graph convolution layer. Another distinction is *L2* regularization being applied to all weight matrices.

Our work is unique from the above works based on the facts that we have used large-scale networks in our experiments and have not limited the number of communities as done in [10], [23]. We also compare our work with GraphSAGE [10] and our model shows better performance (as presented in Section V-D). We have done a comprehensive experimentation taking networks from diverse domains whereas only small-sized citation networks were used in previous work [9]. Thus, our method is not prone to any bias that a particular domain or class of dataset might introduce. We have also used vertex-based metrics to generate Graph Feature Set and have made prediction on this feature set as well. In addition, we have made our model scalable to large graphs and compute systems using GPUs.

Cluster-GCN [26] is a SGD-based (Stochastic Gradient Descent) algorithm to design the minibatches based on efficient graph clustering algorithms. This method restricts the neighborhood search within the subgraph and achieves good memory and computational efficiency. GraphSAINT [27] is another graph sampling based inductive learning method that constructs minibatches by sampling the training graph, rather than the nodes or edges across GCN layers. This method ensures extracting appropriately connected subgraphs so that little information is lost when propagating within the subgraphs. In this method information of many subgraphs are combined together so that the training process overall learns good representation of the full graph. Our work differs from both in choosing the minibatches. In our work we do the sampling based on dense neighborhoods and keep track of the nodes in two consecutive layers. In our sample networks given in the experiments, the communities or labels vary within a large range, but these two methods experiment on networks with a limited number of labels.

III. Preliminaries

In this section, we present a brief introduction to Graph Convolutional Network (GCN). GCN is a neural network that operates on graphs. GCN utilizes the graph structure and collects node information from the neighborhoods in a convolutional manner. Given a graph $G = (V, E)$, a GCN takes as input

- an input feature matrix, X of $N \times F$ dimension, where, $N =$ number of nodes in graph G and $F =$ number of input features for each node, and
- an adjacency matrix, A $(N \times N)$ of graph G.

There is a hidden layer $H^i = f(H^{i-1}, A))$ where $H^0 = X$ and f is a propagation rule. Each layer H^i corresponds to an $N \times F^i$ feature matrix. Features are aggregated to form the next layer's features using the propagation rule f. The output label is denoted by Y.

GCN approaches fall into the following two categories:

- Spatial-based: Feature information from local neighbors are aggregated.

- Recurrent-based: Steady states of nodes are collected.
- Composition-based: Higher orders of neighborhood information are obtained.

■ Spectral-based: Noise from graph signals are removed. An Eigen decomposition of the Laplacian Matrix of the graph is performed.

Here we will focus on Spectral GCNs only. For an L-layer GCN, the layer-wise propagation rule can be written as Eqn. 1.

$$Z^{(l+1)} = A' X^{(l)} W^{(l)}, X^{(l+1)} = \sigma(Z^{(l+1)}), \quad (1)$$

where, $X^{(0)} = X$ and $Z^{(L)} = Y$. The activation function $\sigma(.)$ is often the element-wise ReLU function. Weight Matrix $W^{(l)} \in \mathbb{R}^{F \times F}$ is the feature transformation matrix. A' is the normalized and regularized adjacency matrix. For semi-supervised node classification, we have to minimize the error over all labeled examples. The error function is given in Eqn. 2. We use *Cross-entropy Error* as the loss function.

$$\mathcal{L} = \frac{1}{\mid Y \mid} \sum_{i \in Y} \text{loss}(Y_i, z_i^L) = -\frac{1}{\mid Y \mid} \sum_{i \in Y} \sum_{c=1}^{M} Y_{i,c} \ln p_{i,c} \quad (2)$$

Here, M is number of node-labels/class. Y_i is predicted probability observation. $p_{i,c}$ denotes Y_i is of class c.

IV. Methodology

In this section, we describe our machine learning model by presenting the methods for feature generation, training, testing and validation.

A. Computational Tools/Libraries

We use PyTorch [28] for deep learning models. Initially, we use Gephi [29] for feature generation from node statistics of the graph. Subsequently, we use GraphVis: Interactive Visual Graph Mining and Machine Learning Tool [30] to export node-based features from the graphs. We also use Weka [31] for selecting attributes after generating node-based feature set. Pytorch 1.0.1 has been installed within Anaconda [32] 4.6.11 environment. Python scikit-learn [33], [34] module is used for performance evaluation of the model.

B. Model Classifier

We follow two steps to build our classifier. In the first step, we generate features from the datasets. We train the model using the generated features in the second step. Both of these steps are described in the subsequent sections.

1) Feature Generation: We have used two types of feature sets. One feature set is Identity Matrix where the model is aware of the identity of each node by a unique one-hot vector. The identity matrix is of $N \times N$ dimension, where, N is the number of nodes. A one-hot vector is a $1 \times N$ matrix (vector), being used to represent the encoding of each node in the graph. The vector consists of 0s in all cells with the exception of a single 1 in a cell used uniquely to identify each node. Using the identity matrix as the feature matrix results in highly local representations of each node, i.e., nodes that belong to the same area of the graph are likely to be embedded closely together. In case of distant areas it is difficult for the network to share knowledge in an inductive fashion. In spite of this difficulty, results are comparable to other embeddings using more expensive unsupervised training procedure.

TABLE I: Node-based Feature Set Generated Using Gephi

No.	Attribute	Description
1	Degree	the number of edges connected to the node
2	Triangle	the number of times a node is included in forming a triangle
3	Clustering Coefficient (Watts-Strogatz)	a measure of how complete the neighborhood of a node is
4	Betweenness Centrality	measures how often a node appears on shortest paths between nodes in the network
5	Bridging Coefficient	the average probability of leaving the direct neighbor subgraph of a node
6	Bridging Centrality	a node centrality index based on information flow and topological locality in networks; product of the betweeness centrality and the bridging coefficient
7	Eccentricity	the maximum graph distance between a node and any other vertex of the graph
8	Eigen Centrality	a measure of node importance in a network based on a node's connections
9	Pagerank	measures the importance of each node within the network
10	Authority	indicates the value of the node itself
11	Hub	estimates the value of the links outgoing from the node

Another feature set is based on node statistics of the network. We use Gephi to generate this feature set. The attributes for this feature set is shown in Table I. After generating the features we normalize the features using Weka to keep consistency among the values. While working with larger datasets, we cannot use Gephi to get the node statistics because of graph size limitations in Gephi. Gephi can support graphs upto 1 million nodes and 1 million edges. As we are experimenting with graphs larger than that size, to generate the node statistics we use another tool, *GraphVis*: Interactive Visual Graph Mining and Machine Learning Tool. Another reason to use GraphVis is that using the features generated by GraphVis show better accuracy compared to that of Gephi. For GraphVis, we use the features numbered 1, 2, 3, 4, 7, and 9 from Table I, same as Gephi. Additional features generated by GraphVis is given in Table II.

TABLE II: Additional Node-based Features Generated Using GraphVis

No.	Attribute	Description
a	kcore-number	The core number of a node is the largest value k of a k-core containing that node.
b	4-clique	Number of 4-cliques formed by the vertex
c	4-chordal-cycle	Number of 4-chordal-cycle formed by the vertex
d	4-tailed-triangle	Number of 4-tailed-triangle formed by the vertex

2) Training Model: At first, we apply the Louvain algorithm to our input graph to generate the community of each

node for using as label to train the model. We use a diverse set of data for which ground-truth community is not available. Louvain algorithm is a well-established algorithm and widely used for community detection [1], [7], [17]. It is an efficient heuristic for community detection and used for large-scale datasets using the parallel variants of Louvain method [20], [35], [36]. So, we detect the communities from thlso compares oure graph datasets using Louvain algorithm and use as the ground-truth for each network. The community distribution found from the Louvain algorithm is used for labelling the classes for the labelled dataset. Therefore, the accuracy found in our model a model with the Louvain algorithm. As different networks have different community structure, generating the labelled dataset is necessary for each individual network. If we have sufficient labelled data available for a network, we can skip the initial label generation for the networks using Louvain algorithm.

As we are using semi-supervised classification, we divide our dataset into training, validation and test sets. We keep the ratio of training data lower for semi-supervised learning. Initially, we use a random distribution of training, validation and test sets. We keep the *label rate* 0.001 for the random distribution. Label rate is the ratio of the size of training data to total labelled data. Validation set and test set are divided into $1:2$ ratio within the rest of the labelled data. Random distribution does not work well because many classes might have very few labelled instances and those classes cannot be classified correctly. Thus, we choose classes that have at least 30 instances and discard classes having fewer instances than 30. We name it as '30-I' for 30 instances and 'N-I' in general while considering classes with N instances. Training, validation and test sets are divided similarly as random distribution. The main difference is that instead of the whole labelled dataset, instances are chosen depending on the number of instances in each class. The performance of node-based feature set is poor using '30-I' (we describe the reason in Section V). Therefore, we choose '40-I' for this feature set and get improved accuracy. Based on the number of labelled data using '30-I' and '40-I', we choose the optimal label rate given in Section V.

We apply the GCN Model as described in [9] for training. We use non-linear activation function ReLU [37] for propagation rule. We perform forward propagation through the GCN. We apply the ReLU function row-wise on the last layer in the GCN. We compute the cross entropy loss on known node labels. We backpropagate the loss and update the weight matrices in each layer. The model is trained for a specified number of epochs where the loss is calculated for each training example and the error is backpropagated. But in case of larger networks (networks having more than 70,000 vertices), instead of Batch Gradient Descent, we choose Mini-batch Gradient Descent. Batch size of 1200 is used throughout the experimentation. We sample the mini batches by selecting the nodes with dense neighborhoods. We keep the track of the nodes in consecutive two layers so that our algorithm does not suffer from sparse connection problem. Our Mini-batch GCN algorithm is presented in Algorithm 1. The next step is then invoked to update the model parameters. We train our model to a maximum number of epoch and use early stopping with a window size of 10. We stop training if the validation loss does not decrease for 10 consecutive epochs. We tune the hyper parameters of the model to get the best performance. The optimized parameters are described in Section V. These parameters are dependent on the nature and properties of the network.

Algorithm 1: GCN using Mini-batch Gradient Descent

Input: Graph Adjacency Matrix A, Feature Matrix X, Label Y, Layer Depth K;
Output: Node representation Z

```
Partition graph nodes into b mini-batches V_1, V_2, ..., V_B
for k = 1...K do
    for b = 1...B do
        for u ∈ V_b do
            Update Weight matrix W^(k)
        end
    end
    Calculate Z
end
```

To speed up the execution of our model, we use the functionality of PyTorch on GPUs using CUDA. Pytorch uses one GPU by default and we use multiple GPUs by running our model parallely using *DataParallel* approach. *DataParallel* automatically splits data and sends job orders to multiple models on multiple GPUs. After each model finishes their job, *DataParallel* collects and merges the results before returning. As available GPU memory is limited to 5.566GB in QB2, while working with the larger networks, we load a chunk (portion) of data since the entire graph does not fit in memory for computation. We keep the chunk size 1200 in our experimentation. Therefore, for our largest (according to number of vertices) graph [DBLP (317,080 vertices)], around 1.37GB space is taken by each of the adjacency matrix and the identity matrix. So, for a single pass of the full network, it takes around 70,000 iterations for loading the data into memory for further computations. We also change our model from Batch Gradient Descent to Mini-batch Gradient Descent in GPU because full training set is not available to the model for larger datasets. Thus, the weights are updated using the mean gradient of the mini-batch.

V. Experimental Evaluation

In this section, we summarize the results from our experimentation. We describe the datasets used in our experimentation in Sec. V-A, and report the computational setting in Sec. V-B we have used for this paper. In Sec. V-C, we describe the results to optimize our model with minimized error. We compare our model with the state-of-the-art methods in Sec. V-D. Finally, in Sec. V-E, we show the performance of our model in different computing environment.

TABLE III: Networks Used in Experimentation

Network Type	Name	Vertices	Edges	Description	Source
Citation	cit-DBLP	12,591	49,743	DBLP Citation Network	[30]
	cit-hepTh	27,770	352,807	Arxiv High Energy Physics Theory paper citation network	[6]
	cit-hepPh	34,546	421,578	Arxiv High Energy Physics Phenomology paper citation network	[6]
Communication	email-Eu-core	1,005	25,571	Email data from a large European research institution	[6]
	email-Enron	36,692	183,831	Email communication network from Enron	[6]
	email-dnc	1,900	37,400	Network of emails in the 2016 Democratic National Committee email leak.	[30]
Internet	p2p-Gnutella31	62,586	147,892	Gnutella peer to peer network from August 31 2002	[6]
	p2p-Gnutella24	26,518	65,369	Gnutella peer to peer network from August 24 2002	[6]
	p2p-Gnutella08	6,301	20,777	Gnutella peer to peer network from August 8 2002	[6]
Collaboration	DBLP	317,080	1,049,866	DBLP collaboration network	[6]
	ca-GrQc	5,242	14,496	Collaboration network of Arxiv General Relativity	[6]
	ca-HepPh	12,008	118,521	Collaboration network of Arxiv High Energy Physics	[6]
Biological	human-BNU	177,679	15,669,037	Human brain network where edges represent fiber tracts that connect one vertex to another	[30]
	bio-grid-human	9,527	62,364	category of Biological Networks	[30]
	bio-human-gene-2	22,283	12,345,963	Human gene regulatory network derived from analyzing gene expression profiles.	[30]
Road	road-euroroad	1,174	1,417	Europe Road Network	[30]
	road-usroads	129,164	165,435	US road network	[30]
	road-luxembourg-osm	114,599	119,666	Open Street Map Road Network of Belgium	[30]
Social	slash-dot	77,360	905,468	Slashdot social network, containing friend/foe links between the users of Slashdot, obtained in November 2008	[6]
	wiki-Vote	7,115	103,689	Wikipedia who-votes-on-whom network	[6]
	ego-Facebook	4,039	88,234	Social circles from Facebook (anonymized)	[6]

TABLE IV: Performance Evaluation for Different Networks Based on Identity Matrix and Node Features

No.	Network	Accuracy		Precision		Recall		F1 Score	
		Identity Matrix	Graph Feature	Identity Matrix	Graph Feature	Identity Matrix	Graph Feature	Identity Matrix	Graph Feature
1	**cit-DBLP**	85.44	69.17	0.81	0.61	0.87	0.54	0.84	0.57
2	**cit-hepTh**	84.54	57.93	0.83	0.53	0.81	0.64	0.82	0.58
3	**cit-hepPh**	82.80	65.60	0.80	0.66	0.87	0.50	0.83	0.57
4	**email-Eu-core**	86.40	60.24	0.85	0.67	0.82	0.54	0.83	0.60
5	**email-Enron**	82.39	56.54	0.86	0.55	0.75	0.55	0.80	0.55
6	**email-dnc**	85.49	60.72	0.84	0.64	0.75	0.65	0.80	0.65
7	**p2p-Gnutella31**	86.35	61.30	0.86	0.65	0.79	0.50	0.83	0.57
8	**p2p-Gnutella24**	84.61	55.55	0.85	0.53	0.80	0.68	0.82	0.60
9	**p2p-Gnutella08**	82.39	62.50	0.82	0.64	0.87	0.65	0.84	0.65
10	**DBLP**	82.10	69.02	0.85	0.54	0.77	0.79	0.81	0.64
11	**ca-GrQc**	85.80	61.58	0.84	0.51	0.87	0.79	0.85	0.62
12	**ca-HepPh**	82.95	67.74	0.82	0.63	0.87	0.79	0.84	0.70
13	**human-BNU**	85.10	60.97	0.81	0.58	0.76	0.70	0.79	0.63
14	**bio-grid-human**	83.20	68.16	0.86	0.61	0.79	0.66	0.82	0.63
15	**bio-human-gene-2**	86.90	55.43	0.85	0.63	0.78	0.67	0.81	0.65
16	**road-euroroad**	83.42	61.96	0.85	0.69	0.77	0.75	0.81	0.72
17	**road-usroads**	81.14	69.81	0.83	0.52	0.81	0.75	0.82	0.61
18	**road-luxembourg-osm**	81.96	59.11	0.86	0.55	0.76	0.75	0.81	0.64
19	**slash-dot**	86.35	65.32	0.84	0.69	0.80	0.65	0.82	0.67
20	**wiki-Vote**	85.26	58.76	0.86	0.60	0.81	0.68	0.83	0.64
21	**ego-Facebook**	83.76	63.85	0.86	0.59	0.81	0.69	0.83	0.64

TABLE V: Label Rate of Different Networks Used in Experiments for Both Identity Matrix Feature Set and Graph Feature Set [I.M.-Identity Matrix, G.-Node-based Graph features]

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
I.M.	0.019	0.015	0.024	0.022	0.010	0.017	0.025	0.023	0.027	0.019	0.013	0.015	0.002	0.007	0.010	0.009	0.007	0.005	0.003	0.009	0.007
G.	0.018	0.011	0.006	0.010	0.011	0.004	0.007	0.021	0.023	0.010	0.016	0.020	0.002	0.006	0.005	0.019	0.016	0.007	0.006	0.024	0.008

A. Dataset

We describe the datasets used in our experiments in Table III. All of these networks are taken from 7 different domains. All of these are real-world networks and have been collected from SNAP [6] and Network Repository [30]. Experimenting on a large range of networks gives us the chance to evaluate our method extensively, on datasets with varied topological and structural characteristics.

B. Experimental setting

Some initial experiments on smaller graphs have been conducted on Ubuntu 16.04 LTS, intel core i7 processor @3.4GHz×8, and 16GB memory. However, we perform all other experiments (including the ones on large graphs) on Louisiana Optical Network Infrastructure (LONI). QB2 [38], a 1.5 Petaflop peak performance cluster containing 504 compute nodes with over 10,000 Intel Xeon processing cores of 2.8 GHz has been used for the experiments performed. QB2 has two 960 NVIDIA Tesla K20x GPUs with 128GB memory. QB2 has RedHat Enterprise Linux 6 Operating System, 56 Gb/sec (FDR) InfiniBand 2:1 oversubscribed mesh, 1 Gb/sec Ethernet management network, 10 Gb/sec and 40 Gb/sec external connectivity. NVIDIA Driver 396.51 has been used with CUDA 10.0.

C. Model Optimization

We summarize our result in Table IV with respect to the model performance evaluation metrics. We consider the modularity-based communities calculated using the Louvain algorithm [1], [7] as ground-truth. Thus, the accuracy of our model compares our algorithm with the Louvain algorithm. In addition to *accuracy*, we also measure precision, recall, and F1 score as these metrics are widely used for multi-class classification problem. Precision is useful when the costs of false positive is high. Recall is useful when there is a high cost associated with false negative. F1 score is a better measure to use if we need to seek a balance between precision and recall and there is an uneven class distribution. We observe that *identity matrix* based feature set performs better than the *node-based* feature set. The reason behind is that the *node-based* features are not directly correlated with the communities. Two nodes connected to each other are not necessarily "close". So, such node-based features are not a good predictor for communities. We get around 87% accuracy, 86% precision, 87% recall and 85% F1 score for *identity matrix* based feature set. In addition, for node-based feature set, we get around 70% accuracy, 69% precision, 79% recall and 72% F1 score. Imbalanced labelled data might be a reason for the reduced accuracy. The number of classes for multi-class classification is very large. It is one of the reasons for reduced performance of the model in terms of accuracy. In order to get the best model, we tuned the hyper-parameters. For brevity, we have shown the experimentation for the identity matrix based feature set only. For node-based feature set, we use the same optimized hyper parameters.

TABLE VI: Performance Evaluation for Different Networks with Random Train-Test Distribution [I.M.-Identity Matrix, G.-Node-based Graph features]

Network	Accuracy		Precision		Recall		F1 Score	
	I.M.	G.	I.M.	G.	I.M.	G.	I.M.	G.
ego-Facebook	55.18	30.33	.45	.19	.29	.15	.35	.27
email-Eu-core	66.26	55.48	.55	.26	.62	.31	.57	.24
ca-GrQc	63.30	13.17	.70	.10	.63	.13	.63	.09

1) Training Data Distribution: Distribution of training set and test set plays an important role to improve the accuracy of the model. Initially, we use a random distribution of training and test set. Random distribution does not work well in this case. Table VI shows that the accuracy is very poor for some of the graphs using both identity matrix feature as well as node-based graph feature.

TABLE VII: Model Performance Improvement Using Changed Number of Labels Per Class for Node-based Graph Features

Network	30-I	40-I
cit-DBLP	47.88	69.17
email-Eu-core	55.45	60.24
p2p-Gnutella08	49.35	62.50
ca-GrQc	25.69	61.58
bio-grid-human	36.76	68.16
road-euroroad	43.86	61.96
ego-Facebook	35.25	63.85

Thus, instead of random distribution, we choose '30-I' described in Section IV. For Identity Matrix Feature Set, the model accuracy shown in Table IV is found using '30-I' classification. In case of Graph Feature set, initially we consider '30-I' but find that accuracy is poor. Later we choose '40-I' and accuracy improves on an average 69% given in Table VII. Choosing '40-I' suffers from the exclusion of many mid-sized communities from the dataset. Results shown in Table IV for node-based graph feature set is given for '40-I' classification. The improved accuracy is in between 55.4% to 69.8% for all networks reflected in Table IV.

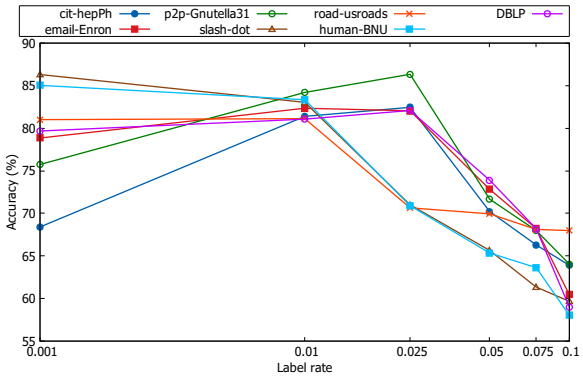
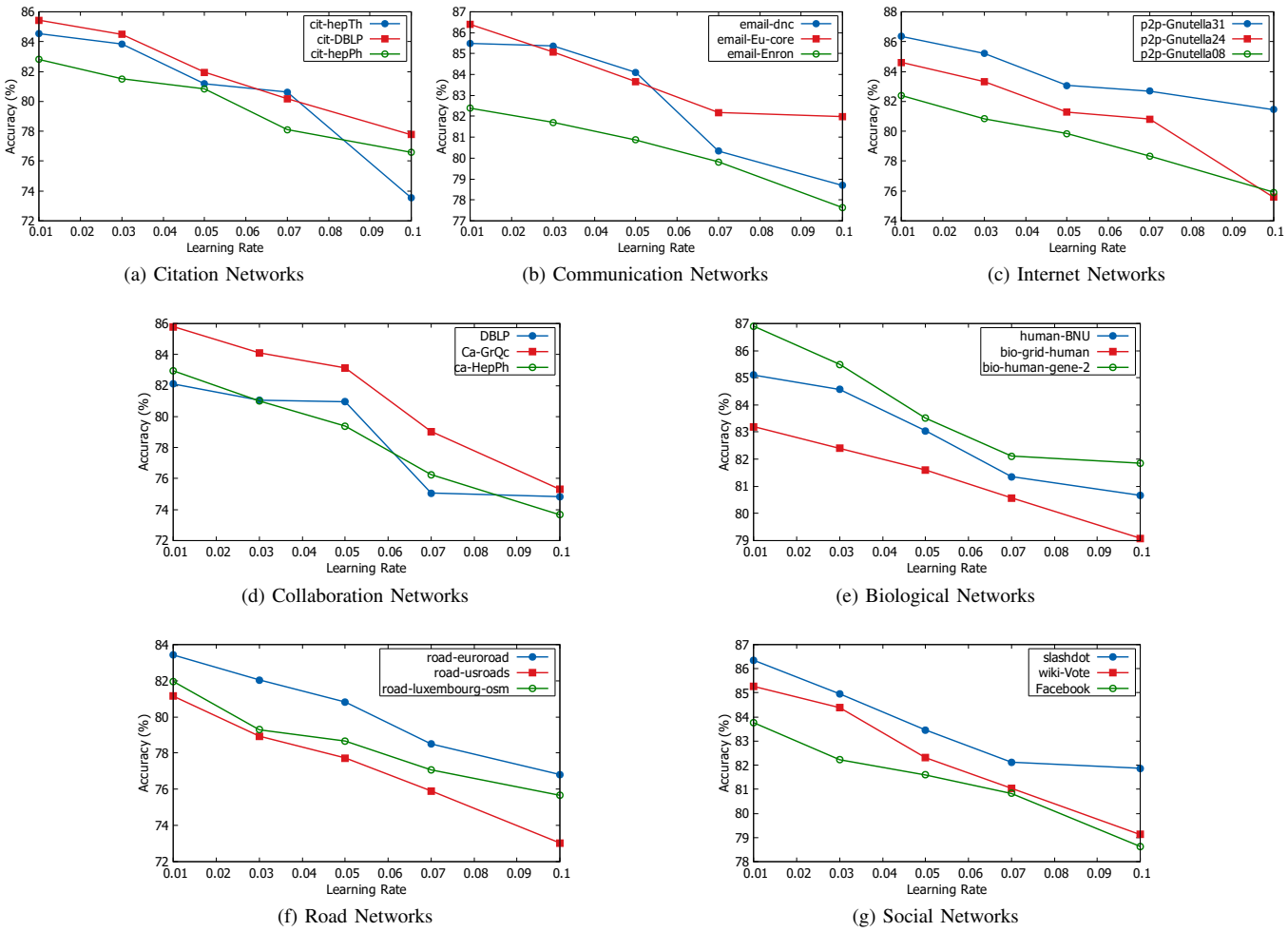


Fig. 1: Change of accuracy with increasing *label rate* for *identity matrix* based feature set. The optimal label rate is in between the range 0.001 to 0.025 for all domains of network, to prevent the model from over-fitting.



(a) Citation Networks (b) Communication Networks (c) Internet Networks

(d) Collaboration Networks (e) Biological Networks

(f) Road Networks (g) Social Networks

Fig. 2: Accuracy for different networks varying learning rate hyper parameter. For learning rate of 0.01, best accuracy is achieved for all network domains.

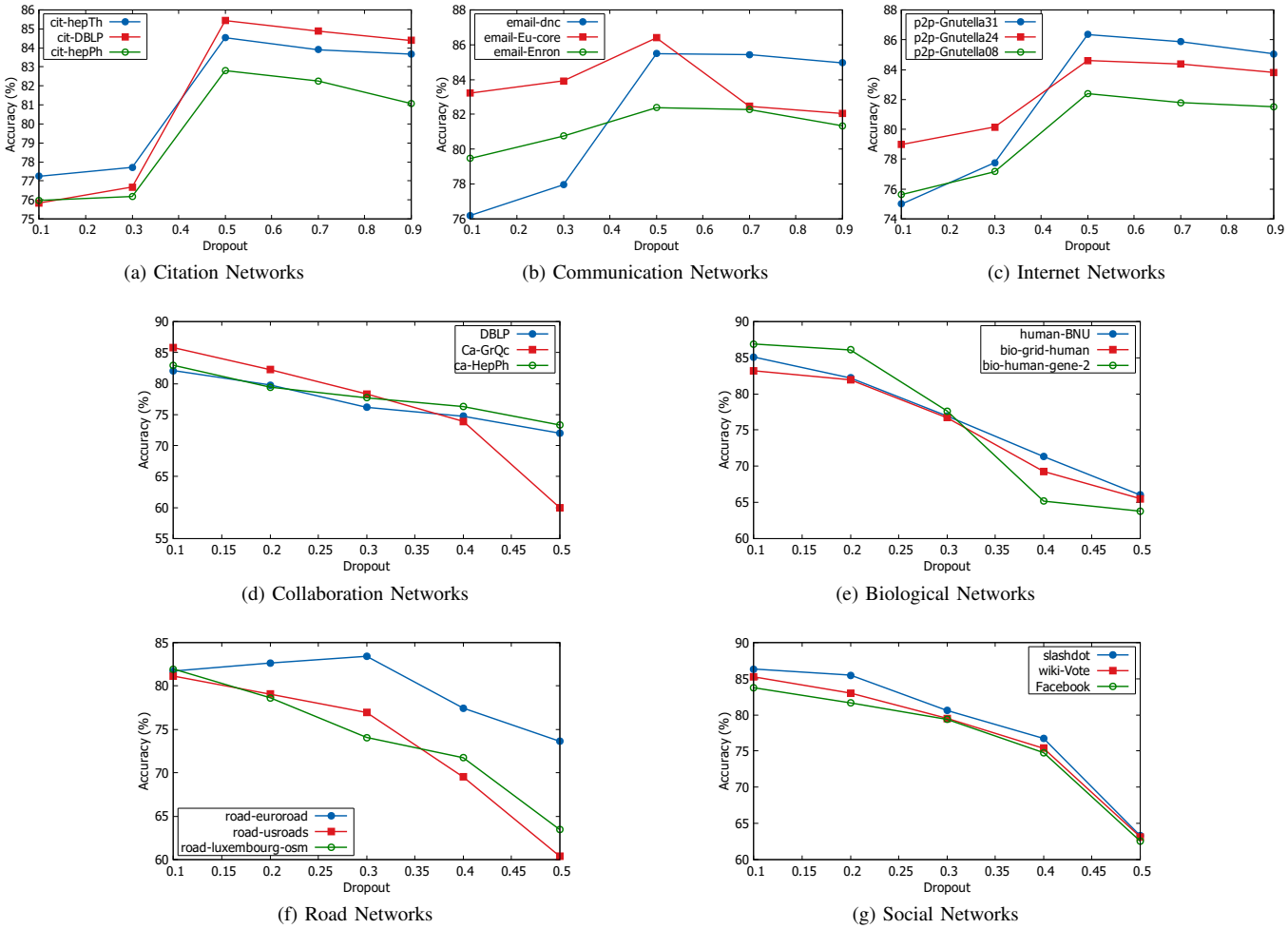
2) Label Rate: Label rate is an important parameter to evaluate the model performance. We keep the label rate small to prevent the model from over-fitting. Figure 1 depicts the change of accuracy with varying label rate for *identity matrix* based feature set. Networks from the same domain show similar behavior. To keep it simple, we have shown a single network from each domain in Figure 1. For social, road and biological network domains, the optimal label rate is in between 0.001 to 0.01. The model tends to over-fit if the label rate goes beyond 0.025 and above. For the rest of the network domains, the optimal label rate is in between 0.01 to 0.025. Based on the optimal values, we use the label rate for each network given in Table V. Number $1-21$ refers to the networks described in Table IV. For node-based graph feature set, we keep the label rate within the range 0.001 to 0.025, to prevent the model from over-fitting.

3) Number of Convolutional Layers: Increasing the number of convolutional layers does not serve our purpose well and the model accuracy decreases by 35.5% on average given in Table VIII. So, we use a 2-layer GCN for our model in all experiments.

TABLE VIII: Change of Model Performance with Varying Number of Convolutional Layers

Network	2 Layers	3 Layers
cit-DBLP	85.44	62.34
email-Eu-core	86.40	59.46
p2p-Gnutella08	82.39	39.54
ca-GrQc	85.80	65.12
bio-grid-human	83.20	35.26
road-euroroad	83.42	55.52
ego-Facebook	83.76	64.34

4) Learning Rate: Figure 2 shows the change of accuracy with learning rate for different types of networks. With smaller learning rate of 0.01, we achieve the best accuracy for each of the networks. We have increased the learning rate up to 0.1 and have not showed results for higher values because the accuracy drops a lot beyond this value. We separately plot each types of networks to understand their pattern. All of these network categories follow a similar pattern. In case



(a) Citation Networks (b) Communication Networks (c) Internet Networks

(d) Collaboration Networks (e) Biological Networks

(f) Road Networks (g) Social Networks

Fig. 3: Accuracy for different networks varying dropout hyper parameter. For Citation, Communication and Internet Networks optimum dropout is 0.5. Except Road Networks, other domains have 0.1 optimum dropout.

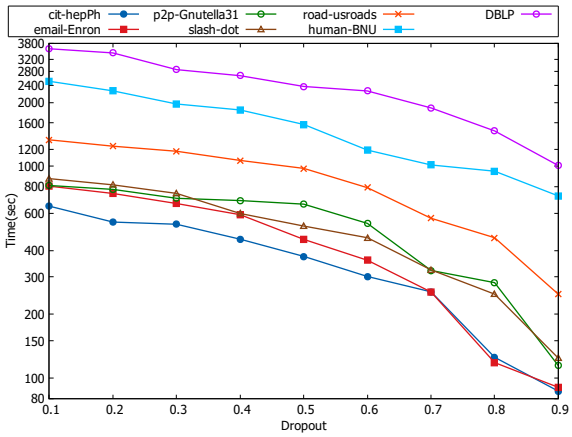
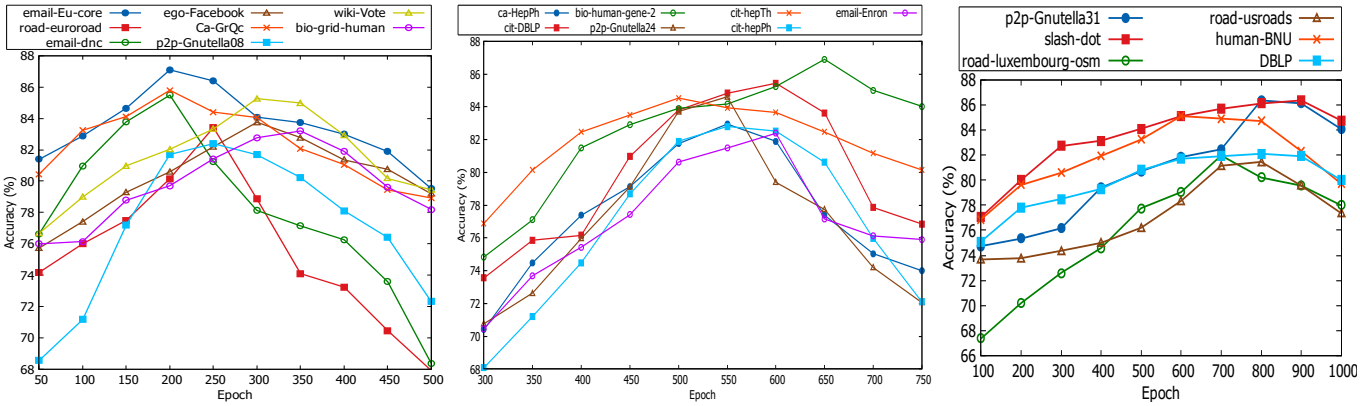


Fig. 4: Time-Efficiency with Increased Dropout Relaxing Accuracy for Networks from All Domains

of biological networks (Figure 2e), after reduced accuracy, the accuracy does not go beyond 80%, and learning rate can be relaxed to 0.1. The change of accuracy is also low and varies within the range of 4 to 5. For communication (Figure 2b) and social (Figure 2g) networks the range of change of accuracy is between 4 to 7. In case of the rest types of networks (Figures 2a, 2c, 2d, 2f), the change of accuracy is higher and varies in between 6 to 11.

5) Dropout: We also vary the dropout given in Figure 3 and determine the optimized parameter with maximum accuracy. For collaboration (Fig. 3d), biological (Fig.3e) and social (Fig. 3g) networks, accuracy decreases with increasing dropout. In case of road networks (Fig.3f), accuracy decreases with dropout. But for "road-euroroad", accuracy is increased up to a certain point, 0.5 then starts decreasing. For the rest three types of networks (Fig.3a, 3b, 3c), accuracy increases with dropout until 0.5, then accuracy falls. Increasing the dropout lessens the run-time of the model shown in 4. In certain cases, where we can relax the accuracy to some extent, we can opt for a higher dropout.



(a) Smaller Networks (b) Medium-sized Networks (c) Larger Networks

Fig. 5: Accuracy for different networks varying the number of epoch. The optimal number of epoch for smaller, medium and larger networks are 200 to 250, 500 to 650 and 700 to 800 respectively.

6) Number of Epoch: We also optimize the number of epoch for each graph that gives the best performance in terms of accuracy, as depicted in Figure 5. From Figure 5a, we observe that for relatively smaller sized networks, the optimum number of epoch is $200 - 250$. In case of the medium sized networks, the epoch size varies in between 500 to 650, as shown in Figure 5b. For larger networks, keeping the epoch in between $700 - 800$ provides the best accuracy, as shown in Figure 5c.

D. Comparison with the Related Methods

TABLE IX: Comparison of Our GCN-based Method and Other Related Existing Methods [10], [39] on *arxiv* and *PPI* datasets.

Method	*arxiv* Dataset [Performance Metric: **Test Accuracy (%)**]	*PPI* Dataset [Performance Metric: **F1 Score**]
MLP	55.50±0.23	N/A
Node2Vec	70.07±0.13	N/A
GCN	71.74±0.29	N/A
GraphSAGE	71.19±0.27	0.612
Our Model	71.43±0.17	0.698

To compare our model with other existing algorithms, we perform experiments on the same datasets the other papers have used. Although our identity matrix based feature set shows better performance, we use the same feature set used by other methods. We do not use identity matrix based feature set to make the comparison consistent and fair. Splitting of the training, validation and test sets are also done in the same manner. *arxiv* is a citation network with $169,343$ nodes and $1,166,243$ edges [39]. Our model shows comparable performance with the base GCN [9] and GraphSAGE [10] shown in Table IX.We also test our model on another dataset *PPI* [10], a biological network with $56,944$ nodes and $818,716$ edges. F1 Score has been reported in [10]. We get better F1 Score given in Table IX. We use Mini-batch Gradient Descent for our model for the performance comparison for both datasets.

E. Model Performance Evaluation

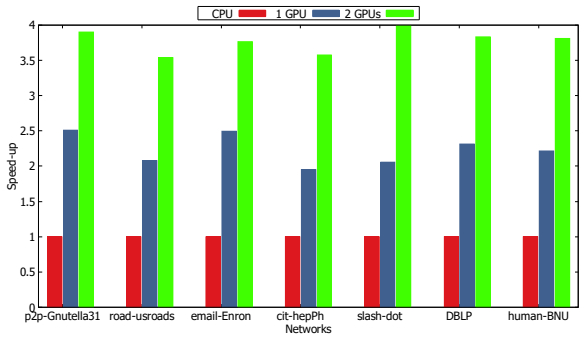


Fig. 6: Speed-up of our model for best performance on GPUs over CPUs. $4\times$ speed-up is observed using 2 GPUs over CPUs

We choose the largest graphs from each types of networks and evaluate their performance in different computing environment. As, node-based graph feature set does not perform better compared to the *identity matrix* feature set, we show the model performance for the identity matrix only. For the largest graph in terms of vertices, with 0.3 Million nodes, the model takes around 3.7 hrs to run on CPUs whereas around 1 hr on GPUs. In case of the largest graph considering edges (15.7 Million), the time required on CPUs is 1.1 hours and 0.7 hours on GPUs. While considering the experiments done on CPUs, we have used a single node of the computing cluster comprising of 20 processing cores. No parallel mechanism has been applied for the CPU computations. From Figure 6 we see that we get around $2.5\times$ speedup using 1 GPU and $4\times$ speedup using 2 GPUs over CPUs.

VI. Conclusion

Semi-supervised learning for community detection is very useful for large-scale networks where detecting communities is computationally expensive and we have access to a few labelled data. Based on the limited information, we can detect

communities of the full network via semi-supervised learning. Our semi-supervised classification model shows an accuracy of up to 86.9% , and 0.85 F1 score for identity matrix based feature set. Node-based graph feature set does not perform equally as the *identity matrix* based feature set but can be used when the memory is limited and the accuracy of the model can be relaxed a bit. We make our model scalable for larger datasets using Mini-batch Gradient Descent resolving the memory issue of GCN. We experiment on 7 different domains of real-world networks and see how our model performs for each of the domains. We have included large-scale datasets in our experiments compared to the smaller datasets used by other related work in literature. We compare our model with the state-of-the-art methods and achieve comparable performance. Our model shows $4\times$ speedup using 2 GPUs over CPUs.

ACKNOWLEDGMENT

This work has been partially supported by Louisiana Board of Regents RCS Grant LEQSF (2017-20)-RDA-25 and University of New Orleans ORSP SCORE award 2019. We also thank the anonymous reviewers for the helpful comments and suggestions to improve this paper.

REFERENCES

[1] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[2] N. S. Sattar and S. Arifuzzaman, "Overcoming mpi communication overhead for distributed community detection," in *Workshop on Software Challenges to Exascale Computing*. Springer, 2018, pp. 77–90.

[3] A. Lancichinetti and S. Fortunato, "Community detection algorithms: a comparative analysis," *Physical review E*, vol. 80, no. 5, p. 056117, 2009.

[4] S. Arifuzzaman and B. Pandey, "Scalable mining, analysis, and visualization of protein-protein interaction networks," *International Journal of Big Data Intelligence (IJBDI)*, vol. 6, no. 3/4, pp. 176–187, 2019. [Online]. Available: https://doi.org/10.1504/IJBDI.2019.100884

[5] S. Arifuzzaman, M. Khan, and M. Marathe, "Fast parallel algorithms for counting and listing triangles in big graphs," *ACM Trans. Knowl. Discov. Data (TKDD)*, vol. 14, no. 1, pp. 5:1–5:34, 2019. [Online]. Available: https://doi.org/10.1145/3365676

[6] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[7] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[8] N. S. Sattar, S. Arifuzzaman, M. F. Zibran, and M. M. Sakib, "Detecting web spam in webgraphs with predictive model analysis," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019.

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

[10] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[11] N. S. Sattar and S. Arifuzzaman, "Data parallel large sparse deep neural network on gpu," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2020, pp. 1–9.

[12] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," in *Advances in neural information processing systems*, 2017, pp. 6530–6539.

[13] S. Rhee, S. Seo, and S. Kim, "Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification," *arXiv preprint arXiv:1711.05859*, 2017.

[14] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 7370–7377.

[15] V. Garcia and J. Bruna, "Few-shot learning with graph neural networks," *arXiv preprint arXiv:1711.04043*, 2017.

[16] X. Wang, Y. Ye, and A. Gupta, "Zero-shot recognition via semantic embeddings and knowledge graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6857–6866.

[17] S. Bhowmick and S. Srinivasan, "A template for parallelizing the louvain method for modularity maximization," in *Dynamics On and Of Complex Networks, Volume 2*. Springer, 2013, pp. 111–124.

[18] C. L. Staudt and H. Meyerhenke, "Engineering parallel algorithms for community detection in massive networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 171–184, 2015.

[19] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical review E*, vol. 70, no. 6, p. 066111, 2004.

[20] N. S. Sattar, "Scalable community detection using distributed louvain algorithm," 2019.

[21] N. S. Sattar and S. Arifuzzaman, "Parallelizing louvain algorithm: distributed memory challenges," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing (DASC)*. IEEE, 2018, pp. 695–701.

[22] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner, "On modularity clustering," *IEEE transactions on knowledge and data engineering*, vol. 20, no. 2, pp. 172–188, 2007.

[23] Z. Chen, X. Li, and J. Bruna, "Supervised community detection with line graph neural networks," *arXiv preprint arXiv:1705.08415*, 2017.

[24] 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," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 249–11 259.

[25] O. Shchur and S. Günnemann, "Overlapping community detection with graph neural networks," *arXiv preprint arXiv:1909.12201*, 2019.

[26] 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 *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 257–266.

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

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[29] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: an open source software for exploring and manipulating networks," in *Third international AAAI conference on weblogs and social media*, 2009.

[30] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: http://networkrepository.com

[31] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[32] "The world's most popular data science platform," =https://www.anaconda.com/.

[33] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler *et al.*, "Api design for machine learning software: experiences from the scikit-learn project," *arXiv preprint arXiv:1309.0238*, 2013.

[34] S.-l. D. Scikit, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[35] M. Halappanavar, H. Lu, A. Kalyanaraman, and A. Tumeo, "Scalable static and dynamic community detection using grappolo," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–6.

[36] S. Ghosh, M. Halappanavar, A. Tumeo, A. Kalyanaraman, H. Lu, D. Chavarria-Miranda, A. Khan, and A. Gebremedhin, "Distributed louvain algorithm for graph community detection," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 885–895.

[37] "Rectifier (neural networks)," =https://en.wikipedia.org/wiki/Rectifier_(neural_networks), Sep 2015.

[38] "Documentation — user guides — qb2," http://www.hpc.lsu.edu/docs/guides.php? system=QB2.

[39] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *arXiv preprint arXiv:2005.00687*, 2020.