Deep Graph Clustering with Random-walk based Scalable Learning

Xiang Li
Computer Science and Engineering
The Ohio State University
li.3880@osu.edu

Dong Li
Computer Science Department
Kent State University
dli12@kent.edu

Ruoming Jin
Computer Science Department
Kent State University
rjin1@kent.edu

Rajiv Ramnath
Computer Science and Engineering
The Ohio State University
ramnath@cse.ohio-state.edu

Gagan Agrawal
Computer and Cyber Sciences
Augusta University
gagrawal@augusta.edu

***Abstract*—Interactions between (social) entities can be frequently represented by an attributed graph, and node clustering in such graphs has received much attention lately. Multiple efforts have successfully applied Graph Convolutional Networks (GCN), though with some limits on accuracy as GCNs have been shown to suffer from over-smoothing issues. Though other methods (particularly those based on Laplacian Smoothing) have reported better accuracy, a fundamental limitation of all the work is a lack of scalability. This paper addresses this open problem by relating the Laplacian smoothing to the Generalized PageRank, and applying a random-walk based algorithm as a scalable graph filter. This forms the basis for our scalable deep clustering algorithm, RwSL. Using 6 real-world datasets and 6 clustering metrics, we show that RwSL achieved improved results over several recent baselines. Most notably, by demonstrating execution of RwSL on a graph with 1.8 billion edges using only a single GPU. We show that RwSL can continue to scale, unlike other existing deep clustering frameworks.**

***Index Terms*—Attributed Graph, Deep Clustering, Laplacian Smoothing, Generalized PageRank, Graph Convolutional Network**

I. Introduction

Many interactions between (social) entities are commonly represented as *attributed graphs* (graphs with *node attributes*). Node clustering [1], when applied to an attributed graph, involves the challenge of jointly learning from the non-Euclidean graph structure and the high-dimensional node attributes. Deep clustering methods [2], which integrate the clustering objective(s) with deep learning (particularly Graph Convolutional Networks (GCNs) [3], [4]), have been investigated by several researchers. A majority of GCN based frameworks for node clustering are based on Graph Autoencoder (GAE) [5]. Especially, advanced clustering performance was achieved from GCN based learning frameworks [6], [7].

However, the introduction of GCN in deep clustering also brought in the inherent limitations of GCN. Previous work has demystified GCN as a special case of Laplacian smoothing – with the implication that it will suffer from the over-smoothing issue which can limit the learning ability [8]. To overcome GCN drawbacks, other efforts use carefully designed graph smoothing filters to ease node clustering tasks, such as those used in Simple Graph Convolution (SGC) [9], Adaptive Graph Convolution (AGC) [10] and Simple Spectral Graph Convolution (SSGC) [11]. It turns out, however, graph smoothing filters based methods still suffer from the same scalability issues (as do the GCN based approaches) since the adjacency matrix of the entire graph is used for a sparse matrix multiplication, resulting in high computational and memory requirements.

To address the open problem of developing scalable and accurate node clustering of an attributed graph, in this paper we derive that the Laplacian smoothing filter [8] is a form of Generalized PageRank (GPR) [12]. Further observing that GPR can be effectively approximated by a scalable and parallelizable random-walk based algorithm [13], we propose a deep clustering framework, **R**andom-**w**alk based **S**calable **L**earning (**RwSL**). To enhance the representative power of input data, we smooth the node attributes with (a parallelizable implementation of) GPR [12], [13] serving as a filter that incorporates the graph topology. We designed a mechanism to integrate an autoencoder with a Deep Neural Networks (DNN) based co-train module, inspired by the recent self-supervised learning paradigm [6], [7], [14].

To summarize, we make the following contributions: 1) We introduce the first deep clustering approach for attributed graphs that can be scaled to an arbitrary sized graph, 2) We analyze spectral properties of a GPR-based graph filter and their influence on clustering performance, 3) We compare our new algorithm with a number of state-of-the-art graph clustering algorithms, and show how our method achieves scalability, while being competitive (or even outperforming the best known methods) with respect to six clustering metrics.

II. Related Work

GAE based methods [5], [15], [16] apply training objectives on reconstructing either adjacency matrix, node attributes, or data representation learned by the DNN. Structural Deep Clustering Network (SDCN) [7] combines an autoencoder module for data representation learning with a GCN module and trains the two deep neural architectures end-to-end

IEEE/ACM ASONAM 2022, November 10-13, 2022
978-1-6654-5661-6/22/$31.00 © 2022 IEEE

for clustering through a self-supervised mechanism. Adaptive Graph Convolution (AGC) [10] is an adaptive graph convolution method for attributed graph clustering that uses high-order graph convolution to capture global cluster structure. Simple Spectral Graph Convolution (SSGC) [11] is a variant of GCN that exploits a modified Markov Diffusion Kernel. Deep Modularity Networks (DMoN) [17] is based on GCN and is an unsupervised pooling method inspired by the modularity measure of clustering quality. The contrastive self-supervised learning based Deep graph Infomax (DGI) [6] incorporates GCN modules and achieves advanced embedding clustering.

III. Methodology

A. Overview

Consider an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$, where $\mathcal{V} = v_1, v_2, \cdots, v_n$ denotes a set of n vertices and d_j denotes the degree of node j. $\mathcal{E}$ is the edge set and $X \in \mathbb{R}^{n \times d}$ is the feature matrix, where each node v is associated with a d-dimensional feature vector X_v. The node clustering task aims to partition the nodes into multiple disjoint groups of similar nodes.

As stated previously, Laplacian smoothing filtering has been used in recent node clustering efforts [9]–[11]. The key foundation of our work is the derivation of the relationship between Laplacian smoothing and Generalized PageRank (GPR). This allows us to perform a novel application of the graph filtering previously proposed (in a different context) by [13], resulting in a filtered attribute matrix $\tilde{X}$ that combines information from both node features and graph topology. If computation of $\tilde{X}$ can be performed on large graphs, the computed values can be fed to DNN based autoencoder. In addition, inspired by the self-supervised learning paradigm, a co-training DNN component is included to further improve the clustering accuracy and performance. Resulting overall method will be presented in Section III-C but is summarized in Figure 1.

B. Laplacian Smoothing Filters and Generalized PageRank

Let A be the adjacency matrix capturing the edges in the graph and D be the diagonal degree matrix. The graph Laplacian matrix is defined as $L = D - A$. Let $\tilde{A} = A + I_n$ denote the augmented adjacency matrix with self-loops added.

The Laplacian smoothing [18] can be written in a matrix form as [8]: $\hat{Y} = (I - \gamma \tilde{D}^{-1}\tilde{L})X$, where $\tilde{D}$ and $\tilde{L}$ ($\tilde{L} = \tilde{D} - \tilde{A}$) are the degree matrix and Laplacian matrix, respectively, corresponding to $\tilde{A}$ defined above. If we replace the normalized Laplacian $\tilde{D}^{-1}\tilde{L}$ with the symmetric normalized Laplacian $\tilde{L}_{sym} = \tilde{D}^{-\frac{1}{2}}\tilde{L}\tilde{D}^{-\frac{1}{2}}$, we will obtain the generalized Laplacian smoothing filter $\mathbf{H}$ [18]:

$$\hat{Y} = HX = (I - \gamma \tilde{L}_{sym})X \tag{1}$$

GCN is a special form of Laplacian smoothing by setting $\gamma = 1$. Previous work [13] proposed a scalable approach, GnnBP (Graph neural network via Bidirectional Propagation), to precompute the feature propagation with the Generalized PageRank matrix (P) [12]. Specifically, in their method:

$$P = S \cdot X = \sum_{l=0}^{L} w_l T^l \cdot X = \sum_{l=0}^{L} w_l (\tilde{D}^{r-1}\tilde{A}\tilde{D}^{r})^l \cdot X \tag{2}$$

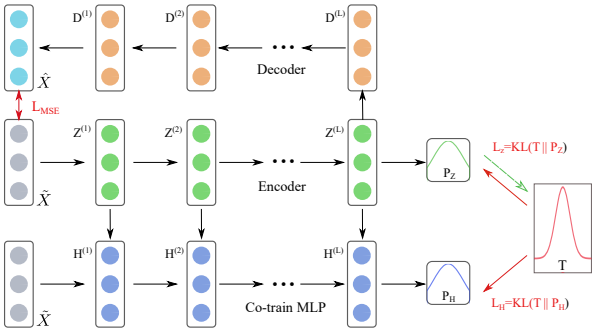


Fig. 1: RwSL framework architecture. $\tilde{X}$ is the input filtered data; $\hat{X}$ is the reconstructed data from the decoder. Z^l and D^l are the latent representation learned from the encoder and decoder. H^l is the latent representation from co-train self-supervised DNN module. P_Z and P_H are respectively the probability distribution of sample assignment to clusters from the last layer of encoder and co-train MLP. The target distribution T, initially calculated from P_Z, will guide the optimization of both auto-encoder and co-train DNN modules.

where $r \in [0, 1]$ is the convolution coefficient, w_l's are the weights of different order convolution matrices satisfying $\sum_{l=0}^{\infty} w_l \leq 1$ and $T^l \cdot X = (\tilde{D}^{r-1}\tilde{A}\tilde{D}^{r})^l \cdot X$ is the l^{th} step propagation matrix (recall that X represents the node attributes). GnnBP [13] was introduced as a localized bidirectional propagation algorithm for an unbiased estimate of the Generalized PageRank matrix (P).

Now, if we stack L Laplacian smoothing filters from Eq.1, we obtain:

$$\begin{aligned} \hat{Y} = H^L X &= (I_n - \gamma(I_n - \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}))^L \cdot X \\ &= (\gamma \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}} + (1-\gamma)I_n)^L \cdot X \end{aligned} \tag{3}$$

The expression obtained above is a special case of the GPR in Eq.2, as can be seen by setting $r = 0.5$ and manipulating the weights w_l to simulate various diffusion processes. Therefore, the Laplacian smoothing filters as used by prior works [9]–[11] can be generalized into GPR – the key foundation of our method and the basis for creating a scalable version.

Computation: Considering both clustering performance and scability, we applied bidirection propagation algorithm [13] as our precomputation procedure. Specifically, we focused on one setup: $w_l = \alpha(1-\alpha)^l$, where $\mathbf{P}$ becomes the PPR (Personalized PageRank) as used in APPNP [19] and $S = \sum_{l=0}^{L} w_l T^l$ corresponds to the PPR diffusion matrix in a previous method [20] (α is the *teleport probability*). The size of the neighborhood from which each node will aggregate information can be controlled by α, i,e., we can balance the needs of preserving locality and leveraging information from a large neighborhood by adjusting α. PPR can even

use infinitely many neighborhood aggregation layers without leading to over-smoothing.

C. Other Details and Overall Algorithm

Auto-encoder Module: With the smoothed attributes $\tilde{X}$ from GPR graph filters as input, we employ a multi-layer perceptron in the auto-encoders network, in which the encoder and the decoder are symmetric. The encoder aims to learn a robust embedding Z^L and the initial clustering is performed using the common K-means algorithm [1]. Following [7], [21], we use student's t-distribution [22] as a kernel to measure the similarity between the representation vector z_i (i-th row of $Z^{(L)}$) and the cluster center vector μ_j for i^{th} sample and j^{th} cluster – more specifically,

$$p_{ij} = \frac{(1 + \|z_i - \mu_j\|^2/v)^{-\frac{v+1}{2}}}{\sum_{j'}(1 + \|z_i - \mu_{j'}\|^2/v)^{-\frac{v+1}{2}}} \tag{4}$$

where v is the degree of freedom of the Student t-distribution. We obtain distribution of sample clustering $P_Z = [p_{ij}]$ by considering p_{ij} as the probability of assigning the sample i to the cluster j. Based on the intuitive motivation to improve the cluster cohesion, a target distribution $T = [t_{ij}]$ can be calculated upon P_Z [7]): $t_{ij} = \frac{p_{ij}^2/f_i}{\sum_{j'} p_{ij'}^2/f_{j'}}$, where, $f_j = \sum_i p_{ij}$ are soft cluster frequencies. Therefore, T will guide the entire model to improve on clustering performance. The decoder will try to reconstruct the input attributes, yielding $\hat{X}$. The preservation of the input information will be tested by checking the reconstruction loss $L_{MSE} = \frac{1}{2N}||\tilde{X} - \hat{X}||_F^2$.

Co-train DNN Module: The complete training process is shown in Algorithm 1. The purpose of the co-train DNN module is to learn the soft probability of assigning samples to separate clusters P_H as from the last DNN layer. Inspired by the self-supervised mechanism [7], we optimize the learning parameters from both DNN and auto-encoder modules simultaneously by minimizing the two KL divergence losses: $L_Z = KL(T\|P_Z)$ and $L_H = KL(T\|P_H)$, together with the reconstruction loss L_{MSE} as on line 14. In the process, we try to preserve the attributed graph information in the auto-encoder embeddings, while continuously enforcing the clusters to be more cohesive. The mini-batch training is applied to ensure a more robust convergence and high scalability.

Algorithm 1 RwSL Training Process

Input data: filtered attributes $\tilde{X}$, graph G, number of clusters: K, maximum iterations *MaxIter*, update period *UpdateIter*

1: Initialize μ with K-means on the representations learned from pre-trained auto-encoder optimized by L_{MSE};
2: **for** $iter \in 0, 1, \vdots, MaxIter$ **do**
3: Generate encoder representations $Z^{(1)}, Z^{(2)}, \vdots, Z^{(L)}$
4: **if** $iter$ mod $UpdateIter == 0$ **then**
5: Use $Z^{(L)}$ to compute distribution P_Z
6: Calculate target distribution T
7: **end if**
8: **for** X_batch in $\tilde{X}$ **do**
9: $H^{(0)} = X\_batch$
10: **for** $l \in 1, \vdots, L-1$ **do**
11: $H^{(l)} = \phi(\tilde{H}^{(l-1)} W_g^{(l)})$
12: $\tilde{H}^{(l)} = (1-\epsilon)H^{(l)} + \epsilon Z^{(l)}$
13: **end for**
14: $H^{(L)} = \phi(\tilde{H}^{(L-1)} W_g^{(L)})$;
15: Feed $Z^{(L)}$ to the decoder to obtain $\hat{X}$;
16: Calculate loss: $L_{tot} = L_{MSE} + \beta L_H + \gamma L_Z$
17: Back propagation and update learnable parameters
18: **end for**
19: **end for**

D. Complexity Analysis

Table I compares the training complexity among different frameworks. RwSL computational cost includes L layers of matrix multiplication for co-train DNN module: $O(L \cdot B \cdot F^2)$ and calculating the target distribution: $O(N \cdot F)$. RwSL memory cost consists of saving a batch of input : $O(B \cdot F)$ and storing learning parameters: $O(L \cdot F^2)$. In contrast, all the other baseline clustering frameworks include an expensive sparse matrix-matrix multiplication of adjacency matrix and feature matrix of the entire graph.

IV. Analysis and Observations

For a clustering task, it is desirable that nodes of the same class have similar feature representations after graph filtering, and we analyze the proposed approach from that angle. As a background [10], [23], each column of a node attribute can be considered as a *graph signal*, which, in turn, can be considered *smooth* if nodes in the neighborhood have similar representations. Furthermore, smooth graph signal should contain more low-frequency than high-frequency basis signals [10].

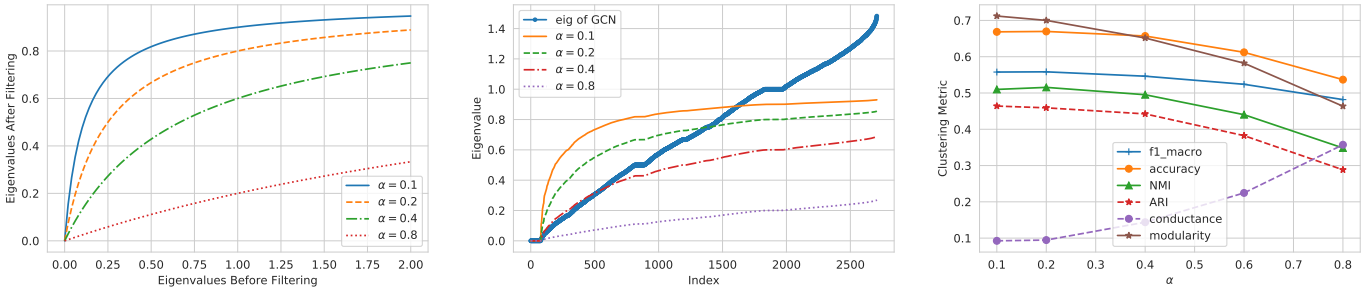
We denote $\lambda_i^{\tilde{L}_{sym}}$ as the eigenvalue of the Laplacian $\tilde{L}_{sym} = I - \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$. Similarly, the eigenvalue of the GCN feature propagation $T = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ is denoted as $\tilde{\lambda}_i^{GCN} = 1 - \lambda_i^{\tilde{L}_{sym}}$ based on $\mathbf{L} = \mathbf{I} - \mathbf{T}$. A K-layer GCN corresponds to filters with the frequency response function $g(\lambda_i^{\tilde{L}_{sym}}) = (1 - \lambda_i^{\tilde{L}_{sym}})^K$ and acts as a low-pass-type filter by taking $K > 1$ [9].

Before presenting our analysis, we consider the following background. It is known that low and high frequencies can capture global and local contexts, respectively, of each node [11]. However, repeatedly applying Laplacian smoothing can lead to *over-smoothing*, where features of nodes within each connected component of the graph will converge to (almost) identical values [8]. Similarly, increasing number of propagation layers can also lead to the filter losing local information associated with the nodes [19], [24].

The PPR based propagation scheme [19] ensures the PageRank score encodes the local neighborhood of root nodes [25] through a controllable parameter *teleporting probability* (α). We investigate the eigenvalues ($\tilde{\lambda}_i^{PPR}$) for the PPR based graph filter with infinite propagation layer (denoted as $\tilde{\lambda}_i^{PPR^\infty}$ after setting $L \to \infty$ in eq.2) which can be transformed to a

TABLE I: Time and Space complexity during Training on the GPU O(·), with N as the number of nodes, E as the number of edges, F as the feature dimension, L as the number of neural network layers, B as the batch size.

Complexity	DGI [6]	SDCN [7]	DMoN [17]	AGC [10]	SSGC [11]	RwSL
Computation Cost	$L \cdot E \cdot F + L \cdot N \cdot F^2$	$L \cdot E \cdot F + L \cdot N \cdot F^2$	$L \cdot E \cdot F + L \cdot N \cdot F^2$	$E \cdot F + N \cdot F$	$E \cdot F + N \cdot F$	$L \cdot B \cdot F^2 + N \cdot F$
Memory Cost	$N \cdot F + E + L \cdot F^2$	$N \cdot F + E + L \cdot F^2$	$N \cdot F + E + L \cdot F^2$	$N \cdot F + E$	$N \cdot F + E$	$B \cdot F + L \cdot F^2$



(a) (b) (c)

Fig. 2: (a) Laplacian eigenvalues of PPR filters $\lambda_i^{\tilde{L}_{PPR}} = 1 - \frac{\alpha}{1-(1-\alpha)(1-\lambda_i)}$ with spectrum $\lambda_i \in [0, 2)$; (b) Sorted by index, Laplacian eigenvalues $\lambda_i^{\tilde{L}_{sym}}$ of a GCN filter and Laplacian eigenvalues $\lambda_i^{\tilde{L}_{PPR}}$ of PPR filters with multiple α on Cora; (c) Teleport probability α influence on clustering metrics for Cora

closed form [20]:

$$\tilde{\lambda}_i^{PPR^\infty} = \alpha \sum_{l=0}^{\infty} ((1-\alpha)\tilde{\lambda}_i^{GCN})^l = \frac{\alpha}{1-(1-\alpha)\tilde{\lambda}_i^{GCN}} \quad (5)$$

Based on Eq. 5, we develop the following claims:

Claim 1. *In the PPR graph filter with infinite propagation,* $S = \sum_{l=0}^{\infty} w_l(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}))^l$ *where* $w_l = \alpha(1-\alpha)^l$*, we have: for* $0 < \alpha_1 < \alpha_2 < 1$*,* $\exists l_0$ *such that* $\forall l > l_0$*,* $w_l(\alpha_1) > w_l(\alpha_2)$*.*

Proof. The weight of each l-hop propagation $w(l) = \alpha(1-\alpha)^l$ for $\alpha \in (0, 1)$ has the following properties:

1) $\sum_{l=0}^{\infty} w_l = \sum_{l=0}^{\infty} \alpha(1-\alpha)^l = 1$,
2) $w(l)$ monotonically decreases with increasing l.

Step 1, we first prove the following fact: $\exists l_0 > 0$ such that $\alpha_1(1-\alpha_1)^{l_0} \geq \alpha_2(1-\alpha_2)^{l_0}$ for $0 < \alpha_1 < \alpha_2 < 1$.

This can be proved by contradiction. Assume, $\forall l > 0$, we have $\alpha_1(1-\alpha_1)^l < \alpha_2(1-\alpha_2)^l$ for $0 < \alpha_1 < \alpha_2 < 1$. Then $\sum_{l=0}^{\infty} \alpha_1(1-\alpha_1)^l < \sum_{l=0}^{\infty} \alpha_2(1-\alpha_2)^l = 1$. This contradicts the fact that $\sum_{l=0}^{\infty} \alpha_1(1-\alpha_1)^l = 1$.

Step 2, $\forall l_1 > l_0$, we have

$$\begin{aligned} \alpha_1(1-\alpha_1)^{l_1} &= \alpha_1(1-\alpha_1)^{l_0}(1-\alpha_1)^{l_1-l_0} \\ &\geq \alpha_2(1-\alpha_2)^{l_0}(1-\alpha_1)^{l_1-l_0} \\ &> \alpha_2(1-\alpha_2)^{l_0}(1-\alpha_2)^{l_1-l_0} = \alpha_2(1-\alpha_2)^{l_1} \end{aligned}$$

□

As indicated by $w(l) = \alpha(1-\alpha)^l$ with $\alpha \in (0, 1)$, the weights decrease exponentially with the propagation distance.

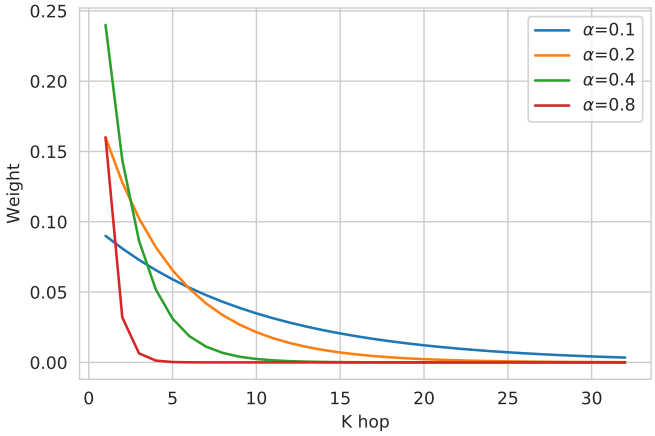


Fig. 3: PPR filter weights w_l for different hops of propagation

A smaller value α will push the distribution of $w(l)$ to weigh more on larger l as in Figure 3.

Thus, a smaller teleport probability $\alpha \in (0, 1)$ indicates a larger receptive field (largest L where $w_L > \epsilon_0$) (for a threshold $\epsilon_0 > 0$), covering L-hop of neighborhood.

Claim 2. *If we denote Laplacian eigenvalues of PPR graph filter with infinite propagation as* $\lambda_i^{\tilde{L}_{PPR}}(\alpha) = 1 - \tilde{\lambda}_i^{PPR^\infty}(\alpha) = 1 - \frac{\alpha}{1-(1-\alpha)(1-\lambda)}$*, where* λ *is a Laplacian eigenvalue, we have* $\forall \lambda \in (0, 2)$*, for* $0 < \alpha_1 < \alpha_2 < 1$*,,* $\lambda_i^{\tilde{L}_{PPR}}(\alpha_1) > \lambda_i^{\tilde{L}_{PPR}}(\alpha_2)$

Proof. With $\lambda \in (0, 2)$ as a constant value, we take the partial derivative of $\lambda_i^{\tilde{L}_{PPR}}(\alpha)$ regarding $\alpha \in (0, 1)$

$$\frac{\partial \lambda_i^{\tilde{L}_{PPR}}(\alpha)}{\partial \alpha} = -\frac{\lambda}{((1-\alpha)\lambda + \alpha)^2} < 0$$

Therefore, for $0 < \alpha_1 < \alpha_2 < 1$, we have $\lambda_i^{\tilde{L}_{PPR}}(\alpha_1) > \lambda_i^{\tilde{L}_{PPR}}(\alpha_2)$

□

Overall, the significance here is that a smaller teleport probability can include information of nodes from further distance (Claim 1), while maintaining node locality by capturing the high frequency signals (Claim 2).

We empirically evaluate the significance of our observations. Figure 2a presents Laplacian eigenvalues of the PPR filter in a typical spectrum range. Figure 2b shows Laplacian eigenvalues of a single GCN layer filter compared with those of PPR filters with multiple α values on the Cora graph. We can observe that the PPR graph filter can preserve the low-frequency of Laplacian and a smaller α can retain more high frequency signals. Finally, we evaluate the influence of α on the clustering performance for Cora in Figure 2c. The inclusion of high-frequency by using small α can capture more information about local neighborhood of each node and yields better performance with respect to clustering metrics. As $w(l) = \alpha(1-\alpha)^l$ with $\alpha \in (0, 1)$ decreases exponentially with the propagation distance l, the PPR filter maintains node locality by weighing more on the nodes in closer neighborhood and thus relieves the over-smoothing issues when including nodes from larger neighborhood.

V. Experimental Results

TABLE II: Datasets Statistics

dataset	Nodes	Classes	Features	Edges
ACM	3025	3	1870	13,128
DBLP	4058	4	334	3528
Citeseer	3327	6	3703	4732
Cora	2708	7	1433	5429
Coauthor CS	18333	15	6805	81,894
Coauthor PHY	34493	5	8415	247,962
Friendster	6.5×10^7	7	40	1.8×10^9

A. Datasets

The RwSL is evaluated and compared with existing state-of-the-art on 6/7 real-world datasets that are summarized in Table II (the seventh dataset, Friendster, could not be used on other methods because of these methods' scalability limitations): **ACM** [7], **DBLP** [7], **Citeseer** and **Cora** [26], **Coauthor CS** and **Coauthor PHY** [27], **Friendster** [28].

B. Experimental Settings

Baselines: We compared the proposed RwSL model with several other (graph) clustering frameworks. **KMeans** [1], a classical clustering method that uses node attributes only; **DeepWalk** [29], an approach for learning latent representations of vertices in a graph using topological information only. Next, we used five most advanced deep clustering frameworks as introduced in Section II: **DGI** [6], **AGC** [10], **SDCN** [7], **DMoN** [17] **SSGC** [11].

Metrics: We employ six popular metrics: accuracy, Normalized Mutual Information (NMI), Average Rand Index (ARI), and macro F1-score are four metrics for ground-truth label analysis, whereas modularity [30] and conductance [28] are graph-level metrics. All metrics except conductance will indicate a better clustering with a larger value.

C. Analysis of Clustering Results - Metrics and Scalability

Table III shows the clustering results on seven datasets – each experiment is performed for 10 times with both the average and range reported. We take two settings for the input of auto-encoder yielding two different versions of our method: 1) using filtered features $\tilde{X}$ with resulting method referred to as **RwSL**; 2) using node features X with resulting version referred to as **RwSL-R**. DGI cannot handle two Coathor datasets due to high GPU memory costs. Our RwSL based models obtained the best results for most clustering metrics for ACM, Citeseer and DBLP while achieving competitive results for most of the rest. RwSL avoids the over-smoothing side-effects of GCN and shows clear advantage in clustering performance over the GCN module based methods such as SDCN [7] and DMoN [17]. Figure 4a shows the training process of RwSL on Coauthor CS. Compared with baseline works with customized graph filters applied such as AGC [10] and SSGC [11], RwSL has additional overhead of frequent data transfers in mini-batches between devices but can achieve higher level of accuracy.

More importantly, none of these baselines can handle large graphs. Figure 4b illustrates the scalability landscape of the state-of-the-art methods. Specifically, we generated synthetic undirected graphs with an increasing number of nodes using PaRMAT [31] with edge size set as 20 times the number of nodes and a random feature matrix with a dimension of 1000. We performed 5-epoch training for each framework. and observed all baselines can only handle graphs with a maximum of hundreds of thousands of nodes, as limited by the GPU memory capacity. In comparison, our proposed method continues to scale linearly. We further performed the node clustering on a billion-scale graph Friendster following the settings as used by authors of GnnBP [13]. Similar settings have been adopted in several prior works on community detection [32]–[34].

D. Node Embedding t-SNE

Figure 5 intuitively shows comparison between node embeddings with node attributes using t-SNE algorithm [35]. We can see that the graph filter and encoder module can gradually ease the clustering tasks by distinguishing clusters with less overlapping areas.

E. Training Convergence

RwSL achieves robust convergence across all real-word datasets as used in this work. For example, Figure 6 shows the training process of RwSL on Coauthor CS with all metrics converged to a steady state.

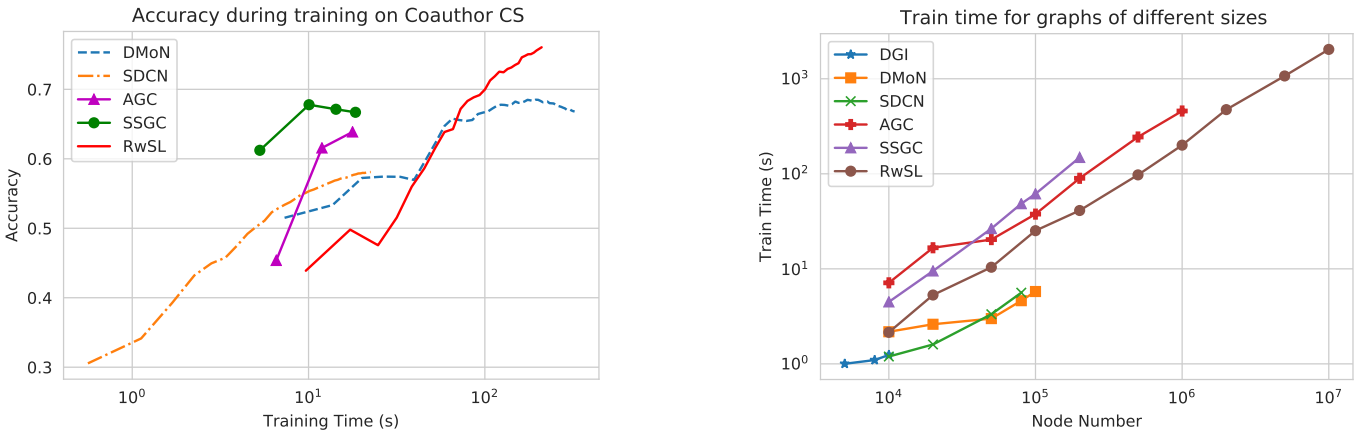
VI. Details for Experimental Setup

A. Experimental Settings

Our RwSL framework is implemented in PyTorch 1.7 on CUDA 10.1, whereas the graph filtering procedure is in C++. Our experiments are performed on nodes with a dual Intel Xeon 6148s @2.4GHz CPU and dual NVIDIA Volta V100 w/ 16GB memory GPU and 384 GB DDR4 memory. Graph filtering is executed on CPU while both autoencoder

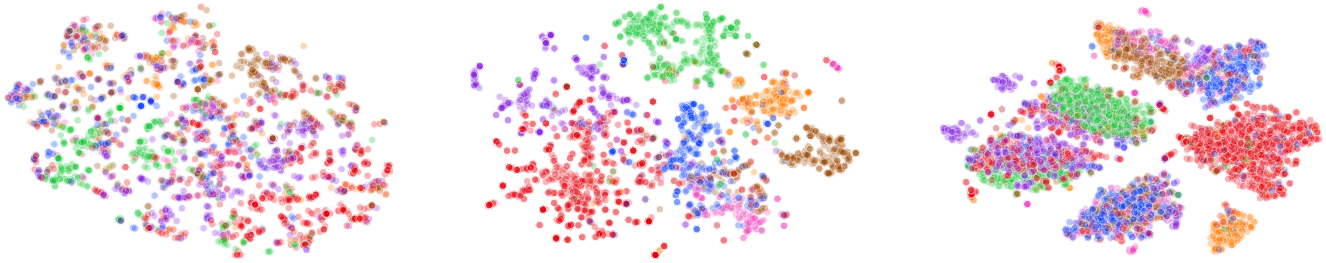
TABLE III: Clustering performance on six datasets (mean±std). RwSL or RwSL-R results highlighted in bold if they have the top 2 clustering performance. The asterisk indicates a convergence issue.

Dataset	Accuracy ↑	NMI ↑	ARI ↑	macro_F1 ↑	Modularity ↑	Conductance ↓
ACM						
KMeans	66.62 ± 0.55	32.41 ± 0.34	30.22 ± 0.41	66.83 ± 0.57	31.20 ± 0.50	30.96 ± 0.23
DeepWalk	50.59 ± 4.27	16.12 ± 4.96	18.56 ± 5.80	46.56 ± 4.43	38.57 ± 9.51	1.79 ± 0.59
SDCN	89.63 ± 0.31	66.74 ± 0.75	72.00 ± 0.75	89.60 ± 0.32	60.86 ± 0.16	3.07 ± 0.17
DMoN	88.51 ± 0.72	63.59 ± 1.49	69.04 ± 1.70	88.57 ± 0.72	61.57 ± 0.17	3.16 ± 0.09
AGC	78.21 ± 0.00	46.31 ± 0.01	48.02 ± 0.00	78.26 ± 0.00	59.44 ± 0.02	2.51 ± 0.01
SSGC	84.43 ± 0.29	56.15 ± 0.51	60.17 ± 0.60	84.44 ± 0.29	60.19 ± 0.05	2.54 ± 0.11
DGI	90.17 ± 0.28	67.84 ± 0.72	73.28 ± 0.66	90.12 ± 0.27	59.79 ± 0.19	3.87 ± 0.14
RwSL	**90.68** ± 0.06	**69.08** ± 0.08	**74.54** ± 0.12	**90.65** ± 0.06	**61.31** ± 0.03	2.99 ± 0.07
RwSL-R	**90.86** ± 0.25	**68.79** ± 0.43	**74.85** ± 0.59	**90.84** ± 0.27	61.29 ± 0.04	3.32 ± 0.21
Cora						
KMeans	35.37 ± 3.72	16.64 ± 4.21	9.31 ± 2.14	31.49 ± 4.58	20.77 ± 3.37	59.77 ± 5.31
DeepWalk	63.87 ± 2.14	44.11 ± 1.33	39.64 ± 1.68	57.98 ± 2.43	72.98 ± 0.79	7.88 ± 0.35
SDCN	64.27 ± 4.87	47.39 ± 3.49	39.72 ± 5.53	57.88 ± 6.99	62.59 ± 5.18	18.32 ± 2.26
DMoN	58.77 ± 6.45	46.10 ± 4.12	37.42 ± 5.27	46.19 ± 7.47	68.70 ± 1.94	8.28 ± 0.74
AGC	65.23 ± 0.93	50.05 ± 0.49	40.23 ± 0.95	58.93 ± 1.68	69.98 ± 0.46	11.08 ± 1.61
SSGC	68.50 ± 1.98	52.80 ± 1.03	45.70 ± 1.28	64.38 ± 2.71	73.71 ± 0.45	9.41 ± 0.55
DGI	68.47 ± 1.43	52.60 ± 0.88	45.63 ± 1.44	65.79 ± 1.53	69.86 ± 0.29	13.64 ± 0.69
RwSL	66.49 ± 0.99	50.90 ± 0.51	45.64 ± 1.91	55.57 ± 0.65	71.09 ± 0.14	**9.17** ± 0.81
RwSL-R	**70.74** ± 0.60	**52.69** ± 0.37	**47.43** ± 1.06	**62.87** ± 0.46	**72.99** ± 0.35	9.33 ± 0.34
Citeseer						
KMeans	46.70 ± 4.33	18.42 ± 3.26	18.42 ± 3.26	44.47 ± 4.44	43.57 ± 2.67	37.21 ± 2.19
DeepWalk	43.56 ± 1.03	16.02 ± 0.56	16.37 ± 0.66	40.37 ± 0.97	76.44 ± 0.20	2.98 ± 0.12
SDCN	63.42 ± 3.31	37.28 ± 2.19	37.40 ± 2.79	56.16 ± 4.53	70.83 ± 2.77	7.98 ± 1.99
DMoN	47.60 ± 3.88	24.86 ± 3.00	22.06 ± 3.12	45.52 ± 3.78	77.06 ± 0.44	5.28 ± 0.66
AGC	67.18 ± 0.52	41.37 ± 0.70	42.10 ± 0.87	62.68 ± 0.48	77.57 ± 0.21	1.72 ± 0.04
SSGC	67.86 ± 0.26	41.86 ± 0.22	42.95 ± 0.30	63.61 ± 0.23	78.03 ± 0.12	1.75 ± 0.03
DGI	68.68 ± 0.76	43.22 ± 0.91	44.53 ± 1.02	64.41 ± 0.70	72.42 ± 0.38	7.19 ± 0.55
RwSL	**70.22** ± 0.07	**44.30** ± 0.15	**46.07** ± 0.16	**66.10** ± 0.09	**78.53** ± 0.04	1.89 ± 0.03
RwSL-R	**69.37** ± 0.18	**44.42** ± 0.20	**45.87** ± 0.23	**65.13** ± 0.18	**78.28** ± 0.07	1.99 ± 0.02
DBLP						
KMeans	38.65 ± 0.58	11.56 ± 0.53	6.95 ± 0.39	31.81 ± 0.53	33.83 ± 0.47	36.20 ± 0.51
DeepWalk	38.99 ± 0.02	5.91 ± 0.02	5.83 ± 0.02	36.87 ± 0.02	64.05 ± 0.03	4.03 ± 0.02
SDCN	69.08 ± 1.95	34.64 ± 1.94	36.31 ± 2.86	67.81 ± 3.46	63.38 ± 1.87	7.56 ± 0.54
DMoN	62.11 ± 6.53	32.54 ± 5.05	33.37 ± 6.43	61.10 ± 6.82	65.63 ± 0.66	6.47 ± 0.54
AGC	69.06 ± 0.06	37.00 ± 0.07	33.69 ± 0.13	68.59 ± 0.05	68.77 ± 0.01	5.29 ± 0.01
SSGC	68.66 ± 1.95	33.89 ± 2.08	37.30 ± 3.13	65.91 ± 2.19	62.02 ± 1.64	3.24 ± 0.52
DGI	59.72 ± 4.68	26.90 ± 4.43	25.12 ± 4.76	59.31 ± 4.69	50.16 ± 3.77	13.84 ± 1.12
RwSL	68.25 ± 0.49	34.39 ± 0.44	34.51 ± 0.76	68.15 ± 0.45	**68.36** ± 0.28	4.13 ± 0.19
RwSL-R	**70.84** ± 0.97	**38.47** ± 0.95	**39.63** ± 1.51	**70.75** ± 0.88	68.34 ± 0.24	4.95 ± 0.07
Coauthor CS						
KMeans	27.96 ± 1.09	15.42 ± 2.25	1.02 ± 0.74	11.68 ± 1.56	9.61 ± 1.88	37.12 ± 4.10
DeepWalk	67.10 ± 2.98	66.67 ± 0.86	53.66 ± 2.91	63.36 ± 2.84	72.88 ± 0.41	17.09 ± 0.66
SDCN	56.86 ± 3.40	54.79 ± 2.44	40.41 ± 4.52	29.36 ± 3.22	53.05 ± 2.02	$^*23.09 \pm 1.89$
DMoN	64.95 ± 3.18	70.06 ± 1.18	60.97 ± 3.23	45.12 ± 2.95	70.79 ± 0.51	15.66 ± 0.71
AGC	62.24 ± 1.81	65.22 ± 0.44	46.96 ± 3.54	51.42 ± 1.27	69.58 ± 0.14	19.80 ± 0.24
SSGC	66.19 ± 1.19	70.06 ± 0.67	58.50 ± 0.17	60.17 ± 1.94	71.82 ± 0.14	19.76 ± 0.22
RwSL	**71.87** ± 3.16	68.44 ± 2.31	**64.73** ± 6.26	49.19 ± 3.03	66.62 ± 1.74	20.88 ± 2.05
RwSL-R	**68.05** ± 1.70	**73.77** ± 0.55	**62.15** ± 0.97	57.68 ± 2.01	71.02 ± 0.47	20.26 ± 0.55
Coauthor PHY						
KMeans	56.19 ± 0.75	11.72 ± 1.92	8.25 ± 1.26	24.74 ± 2.11	5.74 ± 0.83	10.56 ± 1.47
DeepWalk	87.97 ± 0.01	69.13 ± 0.02	79.15 ± 0.03	83.32 ± 0.02	47.96 ± 0.00	5.99 ± 0.00
SDCN	64.65 ± 6.92	50.60 ± 3.71	48.76 ± 9.58	48.51 ± 4.68	44.97 ± 3.16	19.86 ± 7.16
DMoN	67.14 ± 4.74	58.23 ± 5.21	47.72 ± 5.82	59.63 ± 4.85	57.06 ± 2.27	10.89 ± 1.37
AGC	77.41 ± 0.00	62.11 ± 0.02	72.43 ± 0.02	62.09 ± 0.00	45.31 ± 0.00	5.80 ± 0.00
SSGC	55.70 ± 2.26	57.71 ± 1.31	44.91 ± 1.58	55.26 ± 2.30	60.70 ± 0.39	13.47 ± 0.07
RwSL	76.94 ± 0.39	55.60 ± 0.46	64.67 ± 0.73	**64.35** ± 0.48	46.26 ± 0.72	12.93 ± 1.34
RwSL-R	76.93 ± 1.13	**62.46** ± 1.22	68.08 ± 2.52	63.30 ± 0.95	46.24 ± 0.49	9.00 ± 0.63
Friendster						
RwSL	76.49 ± 2.51	44.20 ± 3.73	57.36 ± 4.08	44.65 ± 2.62	6.48 ± 0.16	56.45 ± 2.24



(a) (b)

Fig. 4: (a) Training Process on Coauthor CS; (b) Scalability of Different Frameworks: Training Time vs. Graph Size



(a) Raw node attributes (b) PPR filtered attributes (c) Encoder

Fig. 5: Cora t-SNE for different features/embeddings. Each color represents a distinct class.

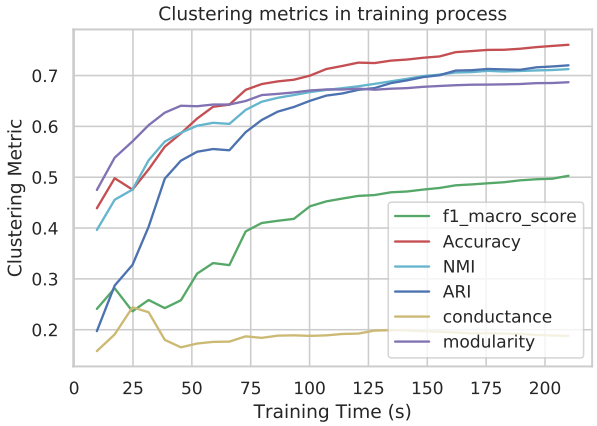


Fig. 6: Clustering Metrics Convergence Curve

pretraining and DNN co-train process are performed on a single GPU. We used dropout to make our model more robust and applied a AdamW optimization method with a decoupled weight decay regularization technique [36].

B. RwSL Hyper-parameter Settings

Detailed hyper-parameters settings are included in Table IV. Learning_rate and pretrain_lr correspond to the learning rates during DNN training and autoencoder training. And n_epochs and pretrain_n_epochs are the number of iterations for DNN training and autoencoder pretrain respectively. The architecture describes the number of neurons on each layer. We set the degree of freedom of the Student t-distribution v as 1.0 globally. β and γ are the weighting factors for the training objective function. We applied a decoupled weight decay regularization [36] resulting in the factor weight_decay and dropout technique with the dropout rate described by dropout_rate. The period (update_p) to update target distribution P is fixed as 1. The last three parameters are from GnnBP framework [37], $\alpha \in (0, 1)$ is teleport probability defined in Personalized PageRank weights $(w_l = \alpha(1-\alpha)^l)$; r_{max} is the threshold during reverse push propagation from the feature vectors; rrz is the convolutional coefficient.

VII. Conclusions

This paper proposed a Random-walk based Scalable Learning (RwSL), which essentially incorporates a Laplacian smoothing based graph filter by using a random-walk based algorithm to approximate Generalized PageRank (GPR). This

TABLE IV: RwSL hyper-parameter settings on 6 datasets

Hyper-parameters	ACM	CiteSeer	DBLP	Cora	Coauthor CS	Coauthor PHY
learning_rate	$1e-4$	$1e-4$	$1e-4$	$1e-4$	$1e-5$	$1e-4$
architecture	512-512-2048-16	512-512-2048-16	512-8	512-2048-32	512-2048-16	512-1024-8
n_epochs	100	120	100	100	30	20
v	1.0	1.0	1.0	1.0	1.0	1.0
β	0.01	0.01	0.01	0.01	0.05	0.05
γ	0.1	0.1	0.1	0.1	0.2	0.2
dropout_rate	0.01	0.01	0.01	0.01	0.01	0.01
weight_decay	0.01	0.01	0.01	0.01	0.5	0.01
update_p	1	1	1	1	1	1
pretrain_lr	$1e-4$	$1e-4$	$1e-4$	$1e-4$	$1e-4$	$1e-4$
pretrain_n_epochs	60	60	60	30	60	30
α	0.1	0.1	0.1	0.1	0.1	0.1
r_{max}	$1e-5$	$1e-5$	$1e-5$	$1e-5$	$1e-5$	$1e-5$
rrz	0.4	0.4	0.4	0.4	0.4	0.4

is made feasible by a novel derivation that establishes a correspondence between Laplacian smoothing and Generalized PageRank (GPR). Our additional theoretical analysis provides spectral properties of a GPR based graph filter and impact of major parameters on clustering performance. Our extensive evaluation has shown that for small and medium sized graphs, we produce results that are better or competitive over the existing methods. Unlike all other methods, we are able to perform deep clustering on a graph with 1.8 billion edges.

Acknowledgements:

This work was partially supported by NSF grants 2007793, 2034850, and 2131509.

REFERENCES

[1] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *JSTOR: Applied Statistics*, 1979.
[2] S. Dong, P. Wang, and K. Abbas. A survey on deep learning and its applications. *Computer Science Review*, 2021.
[3] K. N. Thomas and W. Max. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.
[4] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 2020.
[5] K. N. Thomas and W. Max. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.
[6] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. Hjelm. Deep Graph Infomax. In *ICLR*, 2019.
[7] D. Bo, X. Wang, C. Shi, M. Zhu, E. Lu, and P. Cui. Structural deep clustering network. *WWW*, 2020.
[8] Q. Li, Z. Han, and X. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.
[9] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.
[10] X. Zhang, H. Liu, Q. Li, and X. Wu. Attributed graph clustering via adaptive graph convolution. In *IJCAI*, 2019.
[11] H. Zhu and P. Koniusz. Simple spectral graph convolution. In *ICLR*, 2021.
[12] P. Li, E. Chien, and O. Milenkovic. Optimizing generalized pagerank methods for seed-expansion community detection. *NIPS*, 2019.
[13] M. Chen, Z. Wei, B. Ding, Y. Li, Y. Yuan, X. Du, and J. Wen. Scalable graph neural networks via bidirectional propagation. In *NeurIPS*, 2020.
[14] A. Jaiswal, A.R. Babu, M.Z. Zadeh, D. Banerjee, and F. Makedon. A survey on contrastive self-supervised learning. *Technologies*, 9(1), 2021.
[15] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang. Adversarially regularized graph autoencoder for graph embedding. In *IJCAI*, 2018.
[16] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang. Mgae: marginalized graph autoencoder for graph clustering. In *CIKM*, 2017.
[17] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller. Graph clustering with graph neural networks, 2020.
[18] Gabriel Taubin. A signal processing approach to fair surface design. In *SIGGRAPH*, 1995.
[19] J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2019.
[20] J. Klicpera, S. Weißenberger, and S. Günnemann. Diffusion improves graph learning. In *NeurIPS*, 2019.
[21] C. Wang, S. Pan, R. Hu, G. Long, J. Jiang, and C. Zhang. Attributed graph clustering: A deep attentional embedding approach. In *IJCAI*, 2019.
[22] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *JMLR*, 2008.
[23] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.
[24] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, 2018.
[25] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking : Bringing order to the web. In *WWW*, 1999.
[26] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. Technical report, 2008.
[27] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of graph neural network evaluation, 2019.
[28] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, pages 1–8, 2012.
[29] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *KDD*, 2014.
[30] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.
[31] K. Farzad, G. Rajiv, and B. N. Laxmi. Scalable simd-efficient graph processing on gpus. In *PACT*, 2015.
[32] J. Leskovec, K. J. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, 2010.
[33] K. Kloster and D. Gleich. Heat kernel based community detection. In *KDD*, 2014.
[34] R. Yang, X. Xiao, Z. Wei, S. S. Bhowmick, J. Zhao, and R. Li. Efficient estimation of heat kernel pagerank for local clustering. In *SIGMOD*, 2019.
[35] L. Van Der Maaten. Accelerating t-sne using tree-based algorithms. *J. Mach. Learn. Res.*, 2014.
[36] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.
[37] M. Chen, Z. Wei, B. Ding, Y. Li, Y. Yuan, X. Du, and J. Wen. Scalable graph neural networks via bidirectional propagation. In *NeurIPS*, 2020.