Effective and Scalable Clustering on Massive Attributed Graphs

Renchi Yang
Nanyang Technological University
rcyang@ntu.edu.sg

Jieming Shi*
Hong Kong Polytechnic University
jieming.shi@polyu.edu.hk

Yin Yang
Hamad bin Khalifa University
yyang@hbku.edu.qa

Keke Huang
National University of Singapore
kkhuang@nus.edu.sg

Shiqi Zhang
National University of Singapore
s-zhang@comp.nus.edu.sg

Xiaokui Xiao
National University of Singapore
xkxiao@nus.edu.sg

ABSTRACT

Given a graph G where each node is associated with a set of attributes, and a parameter k specifying the number of output clusters, *k-attributed graph clustering* (k-AGC) groups nodes in G into k disjoint clusters, such that nodes within the same cluster share similar topological and attribute characteristics, while those in different clusters are dissimilar. This problem is challenging on massive graphs, *e.g.*, with millions of nodes and billions of attribute values. For such graphs, existing solutions either incur prohibitively high costs, or produce clustering results with compromised quality.

In this paper, we propose ACMin , an efficient approach to k-AGC that yields high-quality clusters with costs linear to the size of the input graph G. The main contributions of ACMin are twofold: (i) a novel formulation of the k-AGC problem based on an *attributed multi-hop conductance* quality measure custom-made for this problem setting, which effectively captures cluster coherence in terms of both topological proximities and attribute similarities, and (ii) a linear-time optimization solver that obtains high quality clusters iteratively, based on efficient matrix operations such as orthogonal iterations, an alternative optimization approach, as well as an initialization technique that significantly speeds up the convergence of ACMin in practice.

Extensive experiments, comparing 11 competitors on 6 real datasets, demonstrate that ACMin consistently outperforms all competitors in terms of result quality measured against ground truth labels, while being up to orders of magnitude faster. In particular, on the Microsoft Academic Knowledge Graph dataset with 265.2 million edges and 1.1 billion attribute values, ACMin outputs high-quality results for 5-AGC within 1.68 hours using a single CPU core, while none of the 11 competitors finish within 3 days.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Information systems** → **Clustering**.

KEYWORDS

attributed graph, graph clustering, random walk

*Corresponding author.

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449875

ACM Reference Format:
Renchi Yang, Jieming Shi, Yin Yang, Keke Huang, Shiqi Zhang, and Xiaokui Xiao. 2021. Effective and Scalable Clustering on Massive Attributed Graphs. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3442381.3449875

1 INTRODUCTION

Node clustering is a fundamental task in graph mining [28, 39, 44, 61], and finds important real-world applications, *e.g.*, community detection in social networks [12], functional cartography of metabolic networks [15], and protein grouping in biological networks [49]. Traditionally, node clustering is done based on the graph topology, *i.e.*, by grouping together well-connected nodes. This approach, however, is often insufficient to obtain high-quality clusters [13, 22], especially when the graph comes with attributes associated to nodes. In such *attributed graphs*, well-connected nodes tend to share similar attributes; meanwhile, nodes with similar attributes are also likely to be well-connected, as observed in [26, 27]. Therefore, to obtain high-quality node clustering, it is important to consider both graph topology and node attributes. The resulting *attributed graph clustering* has use cases such as gene clustering in biological networks [18], group-oriented marketing in communication networks [54], service/app recommendation, and online advertising in social networks [23, 30].

This paper focuses on *k-attributed graph clustering* (k-AGC), which takes as input an attributed graph G and a parameter k, and aims to partition G into k disjoint node clusters $C_1, C_2, \cdots, C_k$, such that the nodes within the same cluster C_i are not only well-connected to each other, but also share similar attribute values, whereas the nodes in different clusters are distant to each other and share less attributes. It is highly challenging to devise a k-AGC algorithm that yields high-quality clusters, especially on massive graphs, *e.g.*, with millions of nodes and billions of attribute values. Most existing solutions (*e.g.*, [2, 7, 10, 29, 33, 36, 37, 42, 45, 52–54, 62, 64, 67, 68]) fail to scale to such large graphs, since they either incur prohibitive computational overhead, or produce clustering results with compromised quality. For instance, a common methodology [7, 10, 36, 67] relies on materializing the attribute similarity between every pair of nodes in the input graph G, and, thus, requires $O(n^2)$ space for n nodes, which is infeasible for a graph with numerous nodes. Methods based on probabilistic models (*e.g.*, [21, 40, 54, 62, 63]) generally require immense costs on large graphs to estimate the likelihood parameters in their respective optimization programs. Among the faster solutions, some (*e.g.*, [7, 33, 37, 42, 45]) reduce the problem to non-attributed graph clustering by re-weighting

each edge (u, v) in G based on the attribute similarity between nodes u and v. This approach, however, ignores attribute similarities between nodes that are not directly connected, and, consequently, suffers from severe result quality degradation. Finally, k-AGC could be done by first applying attributed network embedding to the input graph (*e.g.*, [17, 31, 34, 55–57, 60, 66]) to obtain an embedding vector for each node, and subsequently feeding the resulting embeddings to a non-graph method such as k-Means clustering [19, 41]. This two-stage pipeline leads to sub-optimal result quality, however, since the node embedding methods do not specifically target for graph clustering, as demonstrated in our experiments.

Facing the challenge of k-AGC on massive attributed graphs, we propose ACMin (short for Attributed multi-hop Conductance Minimization), a novel solution that seamlessly incorporates both graph topology and node attributes to identify high-quality clusters, while being highly scalable and efficient on massive graphs with numerous nodes, edges and attributes. Specifically, ACMin computes k-AGC by solving an optimization problem, in which the main objective is formulated based on a novel concept called *average attributed multi-hop conductance*, which is a non-trivial extension to conductance [6, 61], a classic measure of node cluster coherence. The main idea is to map both node relationships (*i.e.*, connections via edges) and similarities (*i.e.*, common attributes) to motions of a random walker. Then, we show that the corresponding concept of conductance in our setting, *i.e.*, attributed multi-hop conductance, is equivalent to the probability that a random walker starting from a node in a cluster (say, C) terminates at any node outside the cluster C. Accordingly, our goal is to identify a node partitioning scheme that minimizes the average attributed multi-hop conductance among all k clusters in the result.

Finding the exact solution to the above optimization problem turns out to be infeasible for large graphs, as we prove its NP-hardness. Hence, ACMin tackles the problem via an approximate solution with space and time costs linear to the size of the input graph. In particular, there are three key techniques in the ACMin algorithm. First, instead of actually sampling random walks, ACMin converts the optimization objective into its equivalent matrix form, and iteratively refines a solution via efficient matrix operations, *i.e.*, orthogonal iterations [43]. Second, the ACMin solver applies an alternative optimization approach and randomized SVD [16] to efficiently generate and refine clustering results. Third, ACMin includes an effective greedy initialization technique that significantly speeds up the convergence of the iterative process in practice.

We formally analyze the asymptotic time and space complexities of ACMin , and evaluate its performance thoroughly by comparing against 11 existing solutions on 6 real datasets. The quality of a clustering method's outputs is evaluated by both (i) comparing them with ground truth labels, and (ii) measuring their attributed multi-hop conductance, which turns out to agree with (i) on all datasets in the experiments. The evaluation results demonstrate that ACMin consistently outperforms its competitors in terms of clustering quality, at a fraction of their costs. In particular, on the *Flickr* dataset, the performance gap between ACMin and the best competitor is as large as 28.6 percentage points, measured as accuracy with respect to ground truth. On the Microsoft Academic Knowledge Graph (*MAG*) dataset with 265.2 million edges and 1.1 billion attribute values, ACMin terminates in 1.68 hours for a 5-AGC task, while none of the 11 competitors finish within 3 days.

The rest of this paper is organized as follows. Section 2 presents our formulation of the k-AGC problem, based on two novel concepts: attributed random walks and attributed multi-hop conductance. Section 3 overviews the proposed solution ACMin and provides the intuitions of the algorithm. Section 4 describes the complete ACMin algorithm and analyzes its asymptotic complexity. Section 5 contains an extensive set of experimental evaluations. Section 6 reviews related work, and Section 7 concludes the paper with future directions.

2 PROBLEM FORMULATION

Section 2.1 provides necessary background and defines common notations. Section 2.2 describes a random walk model that incorporates both topological proximity and attribute similarity information. Section 2.3 defines the novel concept of attributed multi-hop conductance, which forms the basis of the objective function in our k-AGC problem formulation, presented in Section 2.4.

2.1 Preliminaries

Let $G = (V, E_V, R, E_R)$ be an *attributed graph* consisting of a node set V with cardinality n, a set of edges E_V of size m, each connecting two nodes in V, a set of attributes[1] R with cardinality d, and a set of node-attribute associations E_R, where each element is a tuple $(v_i, r_j, w_{i,j})$ signifying that node $v_i \in V$ is directly associated with attribute $r_j \in R$ with a weight $w_{i,j}$. Without loss of generality, we assume that each edge $(v_i, v_j) \in E_V$ is directed; an undirected edge (v_i, v_j) is simply converted to a pair of directed edges with opposing directions (v_i, v_j) and (v_j, v_i). A high-level definition of the k-AGC problem is as follows.

Definition 2.1 (k-Attributed Graph Clustering (k-AGC) [67]). Given an attributed graph G and the number k of clusters, k-AGC aims to partition the node set V of G into disjoint subsets: $C_1, C_2, \cdots, C_k$, such that (i) nodes within the same cluster C_i are close to each other, while nodes between any two clusters C_i, C_j are distant from each other; and (ii) nodes within the same cluster C_i have homogeneous attribute values, while the nodes in different clusters may have diverse attribute values.

Note that the above definition does not include a concrete optimization objective that quantifies node proximity and attribute homogeneity. As explained in Sections 2.2-2.4, the design of effective cluster quality measures is non-trivial, and is a main contribution of this paper. The problem formulation is completed later in Section 2.4 with a novel objective function.

Regarding notations, we denote matrices in bold uppercase, *e.g.*, $\mathbf{M}$. We use $\mathbf{M}[i]$ to denote the i-th row vector of $\mathbf{M}$, and $\mathbf{M}[:, j]$ to denote the j-th column vector of $\mathbf{M}$. In addition, we use $\mathbf{M}[i, j]$ to denote the element at the i-th row and j-th column of $\mathbf{M}$. Given an index set $\mathcal{I}$, we let $\mathbf{M}[\mathcal{I}]$ (resp. $\mathbf{M}[:, \mathcal{I}]$) be the matrix block of $\mathbf{M}$ that contains the row (resp. column) vectors of the indices in $\mathcal{I}$.

Let $\mathbf{A}$ be the adjacency matrix of the input graph G, *i.e.*, $\mathbf{A}[v_i, v_j] = 1$ if $(v_i, v_j) \in E_V$, otherwise $\mathbf{A}[v_i, v_j] = 0$. Let $\mathbf{D}$ be the

[1] Following common practice in the literature [54, 60], we assume that the attributes have already been pre-processed, *e.g.*, categorical attributes such as marital status are one-hot encoded into binary ones.

Table 1: Frequently used notations.

Notation	Description
G=(V, E_V, R, E_R)	A graph G with node set V, edge set E_V, attribute set R, and node-attribute association set E_R.
n, d	The number of nodes (*i.e.*, $\|V\|$) and the number of attributes (*i.e.*, $\|R\|$) in G, respectively.
k	The number of clusters.
A, **D**, **R**	The adjacency, out-degree and attribute matrices of G.
$\mathbf{P}_V$, $\mathbf{P}_R$	The topological transition and attributed transition matrices of G, respectively.
α, β	Stopping and attributed branching probabilities.
S	The attributed random walk probability matrix (see Eq. (2)).
F	The top-k eigenvectors of **S**.
Y, Ψ(Y)	A $k \times n$ node-cluster indicator (*i.e.*, NCI) and the average attributed multi-hop conductance (*i.e.*, AAMC) of **Y** (see Eq. (8)).

diagonal out-degree matrix of G, *i.e.*, $\mathbf{D}[v_i, v_i] = \sum_{v_j \in V} \mathbf{A}[v_i, v_j]$. We define the topological transition matrix of G as $\mathbf{P}_V = \mathbf{D}^{-1}\mathbf{A}$. Furthermore, we define an attribute matrix $\mathbf{R} \in \mathbb{R}^{n \times d}$, such that $\mathbf{R}[v_i, r_j] = w_{i,j}$ is the weight associated with the entry (v_i, r_j, w_{ij}) $\in E_R$. We refer to $\mathbf{R}[v_i]$ as node v_i's *attribute vector*. Also, let $d_{out}(v_i)$ and $d_{in}(v_i)$ represent the out-degree and in-degree of node v_i in G, respectively. Table 1 lists the frequently used notations throughout the paper.

2.2 Attributed Random Walk Model

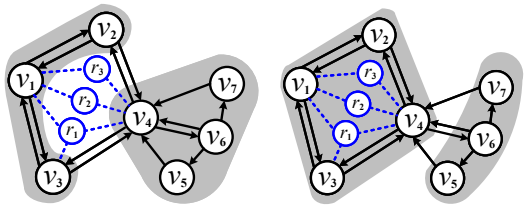
Random walk is an effective model for capturing multi-hop relationships between nodes in a graph [32]. Common definitions of random walk, *e.g.*, random walk with restart (RWR) [25, 47], consider only graph topology but not node attributes. Hence, we devise a new *attributed random walk* model that seamlessly integrates topological proximity and attribute similarity between nodes in a coherent framework, which plays a key role in our formulation of the k-AGC problem, elaborated later.

Given an attributed graph G, we first define the *attributed transition probability* and *topological transition probability* between a pair of nodes v_i and v_j in G. We say that v_i and v_j are connected via attribute r_x, iff. v_i and v_j have a common attribute r_x. For example, in Figure 1, nodes v_1 and v_4 are connected via three attributes $r_1 - r_3$ (shown in blue dashed lines). The attributed transition probability from v_i to v_j via r_x is defined as $\frac{\mathbf{R}[v_i, r_x] \cdot \mathbf{R}[v_j, r_x]}{\sum_{v_l \in V} \sum_{r_y \in R} \mathbf{R}[v_i, r_y] \cdot \mathbf{R}[v_l, r_y]}$, which corresponds to the motion of the random walker that hops from v_i to v_j through a "bridge" r_x. Accordingly, we define the attributed transition probability matrix $\mathbf{P}_R$ of G as:

$$\mathbf{P}_R[v_i, v_j] = \frac{\mathbf{R}[v_i] \cdot \mathbf{R}[v_j]^\top}{\sum_{v_l \in V} \mathbf{R}[v_i] \cdot \mathbf{R}[v_l]^\top}. \quad (1)$$

Intuitively, $\mathbf{P}_R[v_i, v_j]$ models the attributed transition probability from v_i to v_j via any attribute in R.

Meanwhile, following conventional random walk definitions, for any two nodes v_i and v_j that are directly connected by an edge in G, *i.e.*, $(v_i, v_j) \in E_V$, the topological transition probability $\mathbf{P}_V[v_i, v_j]$ from v_i to v_j is $\frac{1}{d_{out}(v_i)}$, where $d_{out}(v_i)$ is the out-degree of node v_i. The topological transition matrix $\mathbf{P}_V$ can then be obtained by $\mathbf{P}_V = \mathbf{D}^{-1}\mathbf{A}$, where **D** and **A** are the node degree and adjacency matrices of G, respectively. Based on the above concepts, we formally define attributed random walk as follows.



(a) Clusters C_1, C_2 **(b) Clusters C'_1, C'_2**

Figure 1: Example attributed graph and clustering schemes.

Definition 2.2 (Attributed Random Walk). Given an attributed graph G, a stopping probability $\alpha \in (0, 1)$, and an attributed branching probability $\beta \in (0, 1)$, an attributed random walk starting from node v_i in G performs one of the following actions at each step:

(1) with probability α, stop at the current node (denoted as v_j),
(2) with probability $1 - \alpha$, jump to another node v_l as follows:
  (a) (attributed transition) with probability β, jump to another node v_l via any attribute with probability $\mathbf{P}_R[v_j, v_l]$,
  (b) (topological transition) with probability $1 - \beta$, jump to an out-neighbor v_l of v_j with probability $\mathbf{P}_V[v_j, v_l]$.

Based on Definition 2.2, the following lemma[2] shows how to directly compute the probability $\mathbf{S}[v_i, v_j]$ that an attributed random walk starting from node v_i stops at node v_j.

Lemma 2.3. *Given an attributed graph G, the probability that an attributed random walk starting from node v_i stops at node v_j is*

$$\mathbf{S}[v_i, v_j] = \alpha \sum_{\ell=0}^{\infty} (1-\alpha)^\ell \cdot ((1-\beta) \cdot \mathbf{P}_V + \beta \cdot \mathbf{P}_R)^\ell [v_i, v_j]. \quad (2)$$

Note that computing **S** directly using Eq. (2) is inefficient, which involves sampling numerous attributed random walks. Instead, the proposed solution ACMin , presented later, computes the probabilities in **S** based on an alternative matrix representation, without simulating any attributed random walk.

2.3 Attributed Multi-Hop Conductance

Conductance is widely used to evaluate the quality of a node cluster in a graph [6, 61]. A smaller conductance indicates a more coherent cluster, and vice versa. Specifically, given a cluster C of graph G, the conductance of C, denoted as $\widehat{\Phi}(C)$, is defined as follows.

$$\widehat{\Phi}(C) = \frac{|\text{cut}(C)|}{\min\{\text{vol}(C), \text{vol}(V \backslash C)\}}, \quad (3)$$

where $\text{vol}(C) = \sum_{v_i \in C} d_{out}(v_i)$, *i.e.*, the sum of the out-degrees of all nodes in C, and $\text{cut}(C) = \{(v_i, v_j) \mid v_i \in C, v_j \in V \backslash C\}$, *i.e.*, the set of outgoing edges with an endpoint in C and the other in $V \backslash C$. Intuitively, $\widehat{\Phi}(C)$ is smaller when C has fewer outgoing edges linking to the nodes outside the cluster (*i.e.*, lower inter-cluster connectivity), and more edges with both endpoints within C (higher intra-cluster connectivity).

In our setting, the classic definition of conductance $\widehat{\Phi}(C)$ is inadequate, since it captures neither attribute information nor multi-hop relationships between nodes. Figure 1 illustrates an example in which $\widehat{\Phi}(C)$ leads to counter-intuitive cluster quality measurements. The example contains nodes v_1-v_7 and attributes r_1-r_3. Suppose that we aim to partition G into two clusters. As shown in Figure 1b, node v_4 is mutually connected to nodes v_2 and v_3, and also shares

[2] All proofs appear in Appendix A

many attributes (*i.e.*, r_1, r_2, and r_3) and neighbors (*i.e.*, v_2 and v_3) with node v_1; in contrast, among nodes v_5-v_7, v_4 is only mutually connected to v_6, and share no common attributes with them. Imagine that this is in a social media setting where each node represents a user, and each edge indicates a follow relationship; then, v_4 is clearly closer to nodes v_1-v_3 than to nodes v_5-v_7, due to its stronger connections and shared attributes to the former group. However, the conductance definition in Eq. (3) leads to the counter-intuitive conclusion that favors the clustering scheme $C_1 = \{v_1, v_2, v_3\}$ and $C_2 = \{v_4, v_5, v_6, v_7\}$ in Figure 1a over C_1' and C_2' in Figure 1b, since the conductance $\widehat{\Phi}(C_1) = \widehat{\Phi}(C_2) = \frac{1}{3} \leq \widehat{\Phi}(C_1') = \widehat{\Phi}(C_2') = \frac{2}{5}$.

To address the above issue, we propose a new measure of cluster quality dubbed *attributed multi-hop conductance*, which can be viewed as an adaptation of conductance to the problem setting of k-AGC. Specifically, given a cluster C of an attributed graph G, suppose that we perform n_r attributed random walks from each node v_i in C. Let $w(v_i, v_j)$ be the number of walks from v_i stopping at v_j. Then, we can use the following quantity instead of Eq. (3) as a measure of cluster coherence:

$$\mathbb{E}\left[\frac{\sum_{v_i \in C, v_j \in V \setminus C} w(v_i, r_j)}{n_r \cdot |C|}\right] = \frac{\sum_{v_i \in C, v_j \in V \setminus C} \mathbb{E}[\frac{w(v_i, v_j)}{n_r}]}{|C|}.$$

Intuitively, the above value quantifies the expected portion of the attributed random walks *escaping* from C, *i.e.*, stopping at any outside node $v_j \in V \setminus C$. Hence, the smaller the number of escaped walks, the higher the cluster coherence. Further, observe that $\mathbb{E}[\frac{w(v_i, v_j)}{n_r}]$ corresponds to the probability that an attributed random walk starting from v_i terminates at v_j, *i.e.*, $\mathbf{S}[v_i, v_j]$ in Eq. (2). Accordingly, we arrive at the following definition of attributed multi-hop conductance $\Phi(C)$.

Definition 2.4 (Attributed Multi-Hop Conductance). Given a cluster C of an attributed graph G, the attributed multi-hop conductance $\Phi(C)$ of the cluster C is defined as

$$\Phi(C) = \sum_{v_i \in C, v_j \in V \setminus C} \frac{\mathbf{S}[v_i, v_j]}{|C|}. \tag{4}$$

2.4 Objective Function

Given an input attributed graph G, we aim to partition all nodes into k disjoint clusters $C_1, C_2, \cdots, C_k$, such that their *average attributed multi-hop conductance* (AAMC) ϕ of the k clusters is minimized, as follows.

$$\phi^* = \min_{C_1, C_2, \cdots, C_k} \frac{\sum_{i=1}^{k} \Phi(C_i)}{k}. \tag{5}$$

The above objective, in combination with Definition 2.1, completes our formuation of the k-AGC problem. As an example, in Figure 1, let $\alpha = 0.2, \beta = 0.5$. Then, we have $\Phi(C_1) = 0.121, \Phi(C_2) = 0.125$ for the clusters C_1, C_2 in Figure 1a, and $\Phi(C_1') = 0.025, \Phi(C_2') = 0.185$ for the clusters C_1', C_2' in Figure 1b. The AAMC values of these two clustering results are $\frac{\Phi(C_1)+\Phi(C_2)}{2} = 0.123 > \frac{\Phi(C_1')+\Phi(C_2')}{2} = 0.105$, which indicate that C_1', C_2' are a better clustering of G, which agrees with our intuition explained in Section 2.3.

3 SOLUTION OVERVIEW

This section provides a high-level overview of the proposed solution ACMin for k-AGC computation, and explains the intuitions behind the algorithm design. The complete ACMin method is elaborated later in Section 4.

First, we transform the optimization objective in Eq. (5) to an equivalent form that is easier to analyze. For this purpose, we introduce the following binary node-cluster indicator (NCI) $\mathbf{Y} \in \mathbb{1}^{k \times n}$ to represent a clustering result:

$$\mathbf{Y}[C_i, v_j] = \begin{cases} 1 & v_j \in C_i, \\ 0 & v_j \in V \setminus C_i, \end{cases} \tag{6}$$

where C_i is the i-th cluster and v_j is the j-th node in the node set V of the input graph G. Based on NCI $\mathbf{Y}$, the following lemma presents an equivalent form of the AAMC objective function in Eq. (5).

Lemma 3.1. *Given a clustering result $C_1, C_2, \cdots, C_k$, represented by NCI $\mathbf{Y}$, the AAMC of $C_1, C_2, \cdots, C_k$ can be obtained by:*

$$\frac{\sum_{i=1}^{k} \Phi(C_i)}{k} = \frac{2}{k} \cdot \mathrm{trace}(((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}) \cdot (\mathbf{I} - \mathbf{S}) \cdot ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})^\top) \tag{7}$$

Then, our optimization objective for k-AMC is transformed to:

$$\phi^* = \min_{\mathbf{Y} \in \mathbb{1}^{k \times n}} \Psi(\mathbf{Y}) \tag{8}$$

where $\Psi(\mathbf{Y}) = \frac{2}{k} \cdot \mathrm{trace}(((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}) \cdot (\mathbf{I} - \mathbf{S}) \cdot ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})^\top)$

Note that Eq. (8) is equivalent to Eq. (5), and yet the former is more friendly to analysis. In particular, we have the following negative result.

Lemma 3.2. *The optimization problem of finding the optimal values of $\mathbf{Y}$ from the objective function in Eq. (8) is NP-hard.* □

Accordingly, to devise a solution for k-AGC on massive graphs, we focus on approximate techniques for optimizing our objective. Observe that the NP-hardness of our objective function in Eq. (8) is due to the requirement that elements of the NCI are binary, *i.e.*, $\mathbf{Y} \in \mathbb{1}^{k \times n}$. Thus, we apply a common trick that relaxes NCI elements from binary to *fractional*, *i.e.*, $\mathbf{Y} \in \mathbb{R}^{k \times n}$. The following lemma shows a sufficient condition to find the optimal values of fractional NCI $\mathbf{Y}$: when the row vectors of $(\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}$ are the top-k eigenvectors of matrix $\mathbf{S}$ (defined in Lemma 2.3), *i.e.*, the k eigenvectors corresponding to the k largest eigenvalues of $\mathbf{S}$.

Lemma 3.3. *Assume that we relax the requirement $\mathbf{Y} \in \mathbb{1}^{k \times n}$ to $\mathbf{Y} \in \mathbb{R}^{k \times n}$. Let $\mathbf{F} \in \mathbb{R}^{k \times n}$ denote the matrix consisting of the top-k eigenvectors of $\mathbf{S}$. Then, the optimal value of $\mathbf{Y}$ for the objective $\min_{\mathbf{Y} \in \mathbb{R}^{k \times n}} \Psi(\mathbf{Y})$ is obtained when $(\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y} = \mathbf{F}$, which leads to a value of $\Psi(\mathbf{Y})$ no larger than the solution of the original optimal objective ϕ^* in Eq. (8).* □

The optimal value of fractional $\mathbf{Y}$, however, does not directly correspond to a clustering solution, which requires the NCI to be binary. The following lemma points to a way to obtain a good approximation of the optimal binary $\mathbf{Y} \in \mathbb{1}^{k \times n}$.

Lemma 3.4. *Given the top-k eigenvectors $\mathbf{F}$ of $\mathbf{S}$, if we obtain a binary NCI $\mathbf{Y}$ that satisfies*

$$\min \|\mathbf{X}\mathbf{F} - (\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}\|_F^2 \quad \text{s.t. } \mathbf{Y} \in \mathbb{1}^{k \times n},\ \mathbf{X}^\top\mathbf{X} = \mathbf{I}, \tag{9}$$

then $\Psi(\mathbf{Y}) \to \phi^*$ in Eq. (8). □

Based on Lemmata 3.3 and 3.4, to approximate the optimal binary NCI $\mathbf{Y}$, we can first compute the top-k eigenvectors $\mathbf{F}$ of $\mathbf{S}$, and then solve for the best NCI $\mathbf{Y}$ that optimizes Eq. (9). The proposed algorithm ACMin follows this two-step approach.

There remain two major challenges in realizing the above idea:

Algorithm 1: ACMin

Input: G, k, α, β.
Output: $\mathbf{Y}$.

1 Compute $\widehat{\mathbf{R}}$ by Eq. (10);
2 $\mathbf{Y}_0 \leftarrow$ InitNCI($\mathbf{P}_V$, $\widehat{\mathbf{R}}$, $\mathbf{R}$, α, β);
3 $\mathbf{F}_0 \leftarrow (\mathbf{Y}_0\mathbf{Y}_0^\top)^{-\frac{1}{2}}\mathbf{Y}_0$;
4 $\mathbf{Y} \leftarrow \mathbf{Y}_0$;
5 $\phi \leftarrow$ AppoxAAMC($\mathbf{P}_V$, $\widehat{\mathbf{R}}$, $\mathbf{R}$, α, β, $\mathbf{Y}_0$);
6 **for** $\ell \leftarrow 1$ *to* t_e **do**
7 $\quad \mathbf{Z}_\ell \leftarrow (1-\beta)\cdot\mathbf{P}_V\mathbf{F}_{\ell-1}^\top + \beta\cdot\widehat{\mathbf{R}}(\mathbf{R}^\top\mathbf{F}_{\ell-1}^\top)$;
8 $\quad \mathbf{F}_\ell \leftarrow$ QR($\mathbf{Z}_\ell$);
9 $\quad$ **if** $\mathbf{F}_\ell = \mathbf{F}_{\ell-1}$ **then break** ;
10 $\quad \mathbf{Y}_\ell \leftarrow$ GenNCI($\mathbf{F}_\ell$);
11 $\quad \phi_\ell \leftarrow$ AppoxAAMC($\mathbf{P}_V$, $\widehat{\mathbf{R}}$, $\mathbf{R}$, α, β, $\mathbf{Y}_\ell$);
12 $\quad$ **if** $\phi_\ell < \phi$ **then** $\phi \leftarrow \phi_\ell$, $\mathbf{Y} \leftarrow \mathbf{Y}_\ell$;
13 **return** $\mathbf{Y}$;

- How to compute $\mathbf{F}$ for large graphs. Note that it is prohibitively expensive to compute $\mathbf{F}$ directly by performing eigen-deomposition on a materialized matrix $\mathbf{S}$ (defined in Eq. (2)), which would consume $\Omega(n^2)$ space and $\Omega(n^2k)$ time.
- Given $\mathbf{F}$, how to efficiently compute $\mathbf{Y} \in \mathbb{R}^{k\times n}$ based on Eq. (9), which in itself is a non-trivial optimization problem.

To address the above challenges, the proposed method ACMin contains three key techniques. First, to compute the top-k eigenvectors $\mathbf{F}$ of $\mathbf{S}$, ACMin employs a scalable, iterative process based on *orthogonal iterations* [43], which does not need to materialize $\mathbf{S}$. Second, to find the best NCI $\mathbf{Y}$, ACMin applies an *alternative optimization approach* and *randomized SVD* [16] to efficiently optimize Eq. (9). Third, to acclerate the above iterative processes, ACMin includes an effective greedy algorithm to compute a high-quality initial value of $\mathbf{Y}$, which significantly speeds up convergence in practice. Overall, ACMin only requires space and time linear to the size of the input graph G. The next section presents the detailed ACMin algorithm and complexity analysis.

4 DETAILED ACMin ALGORITHM

This section presents the detailed ACMin algorithm, shown in Algorithm 1. In the following, Sections 4.1-4.3 detail the three most important components of ACMin : the computation of top-k eigenvectors $\mathbf{F}$, binary NCI $\mathbf{Y}$, and a greedy initialization of $\mathbf{Y}$, respectively. Section 4.4 summarizes the complete ACMin algorithm and analyzes its complexity.

4.1 Computing Top-k Eigenvectors F

Recall from Section 3 that ACMin follows a two-step strategy that first computes $\mathbf{F}$, the top-k eigenvectors of $\mathbf{S}$ (Eq. (2)). Since materializing $\mathbf{S}$ is infeasible on large graphs, this subsection presents our iterative procedure for computing $\mathbf{F}$ without materializing $\mathbf{S}$, which corresponds to Lines 6-9 of Algorithm 1.

First of all, the following lemma reduces the problem of computing $\mathbf{F}$ to computing the top-k eigenvectors of $(1-\beta)\cdot\mathbf{P}_V + \beta\cdot\mathbf{P}_R$.

Lemma 4.1. *Let* $\mathbf{F}$ *be the top-k eigenvectors of* $(1-\beta)\cdot\mathbf{P}_V + \beta\cdot\mathbf{P}_R$. *Then,* $\mathbf{F}$ *is also the top-k eigenvectors of* $\mathbf{S}$. □

Computing the exact top-k eigenvectors of $(1-\beta)\cdot\mathbf{P}_V + \beta\cdot\mathbf{P}_R$ is still rather challenging, however, since materializing $\mathbf{P}_R$ also requires $\Omega(n^2)$ space. To tackle this issue, ACMin applies *orthogonal iterations* [43], as follows. First, ACMin computes a normalized attribute vector $\widehat{\mathbf{R}}[v_i]$ for each node v_i in the graph using the following equation, leading to matrix $\widehat{\mathbf{R}}$ (Line 1 in Algorithm 1).

$$\widehat{\mathbf{R}}[v_i] = \frac{\mathbf{R}[v_i]}{\mathbf{R}[v_i]\cdot\mathbf{r}^\top}\ \forall v_i \in V, \text{where } \mathbf{r} = \textstyle\sum_{v_j\in V}\mathbf{R}[v_j]. \quad (10)$$

Comparing above equation with Eq. (1), it follows that $\mathbf{P}_R = \widehat{\mathbf{R}}\mathbf{R}^\top$. Hence, $(1-\beta)\cdot\mathbf{P}_V + \beta\cdot\mathbf{P}_R$ in Lemma 4.1 can be transformed to $(1-\beta)\cdot\mathbf{P}_V + \beta\cdot\widehat{\mathbf{R}}\mathbf{R}^\top$, eliminating the need to materialize $\mathbf{P}_R$.

Next, suppose that we are currently at the start of the ℓ-th iteration (Line 6 of Algorithm 1) with $\mathbf{F}_{\ell-1}$ obtained in previous iteration. Note that in the first iteration, $\mathbf{F}_0$ is computed from an initial value $\mathbf{Y}_0$ of $\mathbf{Y}$, elaborated in Section 4.3. ACMin computes $\mathbf{Z}_\ell = ((1-\beta)\cdot\mathbf{P}_V + \beta\cdot\widehat{\mathbf{R}}\mathbf{R}^\top)\mathbf{F}_{\ell-1}^\top = (1-\beta)\cdot\mathbf{P}_V\mathbf{F}_{\ell-1}^\top + \beta\cdot\widehat{\mathbf{R}}\cdot(\mathbf{R}^\top\mathbf{F}_{\ell-1}^\top)$ (Line 7 of the algorithm), which can be done in $O(k\cdot(|E_V| + |E_R|))$ time. Then, ACMin employs QR decomposition [9] (Line 8) to decompose $\mathbf{Z}_\ell$ into two matrices: $\mathbf{F}_\ell$ and $\mathbf{\Lambda}_\ell$, such that $\mathbf{Z}_\ell = \mathbf{F}_\ell^\top\cdot\mathbf{\Lambda}_\ell$, where $\mathbf{\Lambda}_\ell$ is an upper-triangular matrix, and $\mathbf{F}_\ell$ is orthogonal (*i.e.*, $\mathbf{F}_\ell\mathbf{F}_\ell^\top = \mathbf{I}$). Clearly, the QR decomposition step can be done in $O(nk^2)$ time, leading to $O(k\cdot(|E_V| + |E_R|) + nk^2)$ total time for one iteration in the computation of $\mathbf{F}$.

Suppose that $\mathbf{F}_\ell$ converges in iteration $\ell = t_c$, *i.e.*, $\mathbf{F}_{t_c}$ is the same as $\mathbf{F}_{t_c-1}$ (Line 9). Then, we have $\mathbf{Z}_\ell = ((1-\beta)\cdot\mathbf{P}_V + \beta\cdot\mathbf{P}_R)\mathbf{F}_{t_c}^\top = \mathbf{F}_{t_c}^\top\cdot\mathbf{\Lambda}_{t_c}$. Considering that $\mathbf{\Lambda}_\ell$ is an upper-triangular matrix, and $\mathbf{F}_\ell$ is orthogonal (*i.e.*, $\mathbf{F}_\ell\mathbf{F}_\ell^\top = \mathbf{I}$), according to [43], we conclude that $\mathbf{F}_{t_c}$ is the top-k eigenvectors of $(1-\beta)\cdot\mathbf{P}_V + \beta\cdot\mathbf{P}_R$ and the diagonal elements of $\mathbf{\Lambda}_{t_c}$ are the top-k eigenvalues. According to Lemma 4.1, the row vectors of $\mathbf{F}_{t_c}$ are also the top-k eigenvectors of $\mathbf{S}$.

Note that throughout the process for computing $\mathbf{F}$, there is no materialization of either $\mathbf{S}$ or $\mathbf{P}_R$, which avoids the corresponding quadratic space requirement. Meanwhile, with a constant k, each iteration takes time linear to the size of the input graph G, which is far more scalable than decomposing $\mathbf{S}$ directly. In practice, the number of required iterations can be significantly reduced through a good initialization, detailed later in Section 4.3.

4.2 Computing Binary NCI Y

As described in Section 3, after obtaining the top-k eigenvectors $\mathbf{F}$ of $\mathbf{S}$, ACMin proceeds to compute the binary NCI $\mathbf{Y}$ by solving the optimization problem in Eq. (9). In Algorithm 1, this is done in Lines 10-12. Note that in ACMin , the computation of $\mathbf{Y}$ is performed once in every iteration for computing $\mathbf{F}$, rather than only once after the final value of $\mathbf{F}$ is obtained. This is because our algorithm is approximate, and, thus, the final value of $\mathbf{F}$ does not necessarily lead to the best clustering quality, measured by AAMC (Section 2.4). Hence, ACMin computes $\mathbf{Y}_\ell$ and the corresponding AAMC ϕ_ℓ for each iteration ℓ, and udpate the current best result $\mathbf{Y}$ and ϕ whenever a better result is found (Lines 11-12 in Algorithm 1).

Next we clarify the GenNCI function, shown in in Algorithm 2, which computes the binary NCI $\mathbf{Y}_\ell \in \mathbb{R}^{k\times n}$ with $\mathbf{F}_\ell$ in the current iteration ℓ. First, based on properties of matrix trace, we transform the optimization objective in Eq. (9), as follows.

$$\|\mathbf{X}\mathbf{F} - (\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}\|_F^2 = 2k - 2\cdot\text{trace}((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}\mathbf{F}^\top\mathbf{X}^\top). \quad (11)$$

Algorithm 2: GenNCI

Input: $\mathbf{F}$.
Output: $\mathbf{Y}$.
1 $\mathbf{X}' \leftarrow \mathbf{I}, \mathbf{X} \leftarrow \mathbf{I}$;
2 **for** $\ell \leftarrow 1$ *to* t_m **do**
3 **for** $i \leftarrow 1$ *to* k **do** Compute γ_i by Eq. (15) ;
4 **for** $v_j \in V$ **do**
5 Pick c_i by Eq. (14);
6 $\mathbf{Y}[:, v_j] \leftarrow \mathbf{0},\ \mathbf{Y}[c_i, v_j] \leftarrow 1$;
7 $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V} \leftarrow \text{SVD}((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}\mathbf{F}^\top)$;
8 $\mathbf{X}' \leftarrow \mathbf{X},\ \mathbf{X} \leftarrow \mathbf{U} \cdot \mathbf{V}^\top$;
9 **if** $\mathbf{X} = \mathbf{X}'$ **then break**;
10 **return** $\mathbf{Y}$;

GenNCI applies an alternative optimization approach to minimize Eq. (11). Specifically, the algorithm updates two variables, $\mathbf{X}$ and $\mathbf{Y}$ in an alternating fashion, each time fixing one of them and updating the other, according to the following rules.

Updating $\mathbf{Y}$ with $\mathbf{X}$ fixed. Given $\mathbf{F}$, according to Eq. (11), with $\mathbf{X}$ fixed, the function to optimize becomes:

$$\max_{\mathbf{Y} \in \mathbb{1}^{K \times n}} \text{trace}((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}\mathbf{F}^\top\mathbf{X}^\top) \tag{12}$$

Let $\mathbf{M} = \mathbf{F}^\top\mathbf{X}^\top$. Eq. (12) is equivalent to

$$\max_{\mathbf{Y} \in \mathbb{1}^{k \times n}} \sum_{v_j \in V} \sum_{i=1}^{k} \left(\mathbf{Y}[c_i, v_j] \cdot \frac{\mathbf{M}[v_j, c_i]}{\sqrt{\sum_{v_l \in V} \mathbf{Y}[c_i, v_l]}} \right). \tag{13}$$

Since $\mathbf{Y} \in \mathbb{1}^{k \times n}$, for each column $\mathbf{Y}[:, v_j]$ ($v_j \in V$), we update the entry at c_i of $\mathbf{Y}[:, v_j]$ (*i.e.*, $\mathbf{Y}[c_i, v_j]$) to 1, and 0 everywhere else, where c_i is picked greedily as follows:

$$c_i = \arg\max_{1 \le c_l \le k} \left[\frac{(1 - \mathbf{Y}[c_l, v_j]) \cdot \mathbf{M}[v_j, c_l]}{\sqrt{\gamma_l^2 + 1}} + \frac{\mathbf{Y}[c_l, v_j] \cdot \mathbf{M}[v_j, c_l]}{\gamma_l} \right], \tag{14}$$

$$\text{where } \gamma_l = \sqrt{\sum_{v_z \in V} \mathbf{Y}[c_l, v_z]}, \tag{15}$$

meaning that we always update each column $\mathbf{Y}[:, v_j]$ ($v_j \in V$) such that the objective function in Eq. (12) is maximized. Since both $\mathbf{M} = \mathbf{F}^\top\mathbf{X}^\top$ and γ_l can be precomputed at the beginning of each iteration, which takes $O(nk^2)$ time, it takes $O(nk)$ time to update the whole $\mathbf{Y}$ in each iteration.

Updating $\mathbf{X}$ with $\mathbf{Y}$ fixed. Given $\mathbf{F}$, according to Eq. (11), with $\mathbf{Y}$ fixed, the function to optimize becomes:

$$\max_{\mathbf{X}^\top\mathbf{X} = \mathbf{I}} \text{trace}((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}\mathbf{F}^\top\mathbf{X}^\top) \tag{16}$$

The following lemma shows that the optimal $\mathbf{X}$ in Eq. (16) can be obtained via singular value decomposition (SVD) of matrix $(\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}\mathbf{F}^\top$.

Lemma 4.2. *The optimal solution to the objective function in Eq.* (16) *is* $\mathbf{X} = \mathbf{U}\mathbf{V}^\top$, *where* $\mathbf{U}$ *and* $\mathbf{V}$ *are the left and right singular vectors of* $(\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}\mathbf{F}^\top$ *respectively.* □

To compute SVD of $(\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}\mathbf{F}^\top \in \mathbb{R}^{k \times k}$, GenNCI employs the randomized SVD algorithm [16], which finishes in $O(k^3)$ time.

With the above update rules for $\mathbf{X}$ and $\mathbf{Y}$ respectively, GenNCI (Algorithm 2) iteratively updates $\mathbf{X}$ and $\mathbf{Y}$ for a maximum of t_m iterations (Lines 2-9). In our experiments, we found that setting t_m to 50 usually leads to satisfactory performance. Note that the iterations may converge earlier than t_m iterations (Line 9). Since updating $\mathbf{Y}$ and $\mathbf{X}$ takes $O(nk^2)$ and $O(k^3)$ time respectively, GenNCI terminates within $O(t_m \cdot (nk^2 + k^3))$ time.

Algorithm 3: InitNCI

Input: $\mathbf{P}_V$, α, β.
Output: $\mathbf{Y}_0$.
1 $\mathbf{Y}_0 \leftarrow \mathbf{0}$, $V_\tau \leftarrow \emptyset$;
2 $V'_\tau \leftarrow \{v_{\tau_1}, v_{\tau_2}, \cdots, v_{\tau_{5k}}\}$ where v_{τ_i} is the node in V with i-th largest in-degree;
3 $\mathbf{\Pi}_0 \leftarrow \mathbf{I}[:, V'_\tau],\ t \leftarrow \frac{1}{\alpha}$;
4 **for** $\ell \leftarrow 1$ *to* t **do** $\mathbf{\Pi}_\ell \leftarrow (1 - \alpha) \cdot \mathbf{P}_V\mathbf{\Pi}_{\ell-1} + \mathbf{\Pi}_0$;
5 $\mathbf{\Pi}_t \leftarrow \alpha \cdot \mathbf{\Pi}_t$;
6 **for** $v_\tau \in V'_\tau$ **do** compute $\sum_{v_j \in V} \mathbf{\Pi}_t[v_j, v_\tau]$;
7 Select the top-k nodes $v_\tau \in V'_\tau$ with the largest $\sum_{v_j \in V} \mathbf{\Pi}_t[v_j, v_\tau]$ into V_τ as the k center nodes;
8 **for** $v_j \in V$ **do** select $v_{\tau_i} \in V_\tau$ with the largest $\mathbf{\Pi}_t[v_j, v_{\tau_i}]$, and set $\mathbf{Y}_0[i, v_j] \leftarrow 1$;
9 **return** $\mathbf{Y}_0$;

4.3 Effective NCI Initialization

Next we clarify the computation of the initial value $\mathbf{Y}_0$ of the NCI (Line 2 of Algorithm 1). If we simply assign random values to elements of $\mathbf{Y}_0$, the iterative process in ACMin from Lines 6 to 12 would converge slowly. To address this issue, we propose an effective greedy initialization technique InitNCI, which usually leads to fast convergence of ACMin in practice, as demonstrated in our experiments in Section 5.4.

Given a cluster C, recall that its attributed multi-hop conductance $\Phi(C)$ (Eq. (4)) is defined based on the intuition that $\Phi(C)$ is lower when an attributed random walk from any nodes in C is more likely to stop at a node within C. Further, we observe that in practice, a high-quality cluster C tends to have high intra-cluster connectivity via certain center nodes within C, and such a center node usually has *high in-degree* (*i.e.*, many in-neighbors). In other words, the nodes belonging to the same cluster tend to have many paths to the center node of the cluster, and consequently, a random walk with restart (RWR) [25, 47] within a cluster is more likely to stop at the center node [46]. Based on these intuitions, we propose to leverage graph topology (*i.e.*, V and E_V of the input attributed graph G) as well as RWR to quickly identify k possible cluster center nodes, $V_\tau = \{v_{\tau_1}, v_{\tau_2}, \cdots, v_{\tau_k}\} \subset V$, and greedily initialize NCI $\mathbf{Y}_0$ by grouping the nodes in V to a center node according to their topological relationships to the center node.

Algorithm 3 presents the pseudo-code of InitNCI. After initializing $\mathbf{Y}_0$ to a $k \times n$ zero matrix and V_τ to an empty set at Line 1, the method first selects from V a candidate set V'_τ of size $5k$ (Line 2), which consists of the top-$(5k)$ nodes with the largest in-degrees. The nodes in V'_τ serve as the candidate nodes for the k center nodes to be detected. Then we compute the t-hop RWR value $\mathbf{\Pi}_t[v_j, v_\tau]$ from every node $v_j \in V$ to every node $v_\tau \in V'_\tau$ from Lines 3 to 5 according to the following equation [59].

$$\mathbf{\Pi}_t = \sum_{\ell=0}^{t} \alpha(1 - \alpha)^\ell \mathbf{P}_V^\ell \cdot \mathbf{I}[:, V'_\tau] \tag{17}$$

Algorithm 4: AppoxAAMC

Input: $\mathbf{P}_V$, $\widehat{\mathbf{R}}$, $\mathbf{R}$, α, β, $\mathbf{Y}$.
Output: ϕ.

1 $\mathbf{H}_0 \leftarrow (\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}$, $t \leftarrow \frac{1}{\alpha}$;
2 **for** $\ell = 1$ *to* t **do**
3 $\quad \mathbf{H}_\ell \leftarrow (1-\alpha)\cdot((1-\beta)\cdot\mathbf{P}_V\mathbf{H}_{\ell-1}^\top + \beta\cdot\widehat{\mathbf{R}}(\mathbf{R}^\top\mathbf{H}_{\ell-1}^\top)) + \mathbf{H}_0$;
4 $\phi \leftarrow \frac{2}{k}\cdot\sum_{i=1}^{k}\mathbf{H}_0[i]\cdot(\mathbf{H}_0^\top[i] - \alpha\cdot\mathbf{H}_t[:,i])$;
5 **return** ϕ;

In particular, we set $t = \frac{1}{\alpha}$ at Line 3, which is the expected length of an RWR, and is usually sufficient for our purpose. If $\mathbf{\Pi}_t[v_j, v_\tau]$ is large, it means that the random walks starting from v_j are more likely to stop at v_τ, which matches our aforementioned intuition of possible cluster center nodes.

Then, at Line 6, for each candidate center node $v_\tau \in V'_\tau$, we compute the sum of $\mathbf{\Pi}_t[v_j, v_\tau]$ from all nodes $v_j \in V$ to v_τ. If v_τ has larger $\sum_{v_j\in V}\mathbf{\Pi}_t[v_j, v_\tau]$, it indicates that the random walks starting from any nodes in V are more likely to stop at v_τ. Therefore, at Line 7, we select the top-k nodes $v_\tau \in V'_\tau$ with the largest $\sum_{v_j\in V}\mathbf{\Pi}_t[v_j, v_{\tau_i}]$ as the k possible center nodes in V_τ. At Line 8, for each node $v_j \in V$, we select the center node $v_{\tau_i} \in V_\tau$ with the largest $\mathbf{\Pi}_t[v_j, v_{\tau_i}]$ and greedily group v_j and v_{τ_i} into the same i-th cluster by setting $\mathbf{Y}_0[i, v_j]$ to 1, completing the computation of $\mathbf{Y}_0$.

Note that Line 2 in Algorithm 3 takes $O(n + k\log(n))$ time, and the computation of $\mathbf{\Pi}_t$ requires $O(\frac{k}{\alpha}\cdot|E_V|)$ time. Therefore, InitNCI runs in $O(\frac{k}{\alpha}\cdot|E_V|)$ time.

4.4 Complete ACMin Algorithm and Analysis

Algorithm 1 summarizes the pseudo-code of ACMin , which takes as input an attributed graph G, the number of clusters k, random walk stopping probability α, and attributed branching probability β (defined in Definition 2.2). Initially (Line 1), ACMin computes matrix $\widehat{\mathbf{R}}$, explained in Section 4.1. Then (Line 2), ACMin computes an initial value $\mathbf{Y}_0$ for $\mathbf{Y}$ via InitNCI (Algorithm 3), and derives the corresponding value $\mathbf{F}_0$ for $\mathbf{F}$ according to Lemma 3.3 in Line 3.

Next (Line 5), we invoke AppoxAAMC (Algorithm 4) that uses $\mathbf{Y}$ to compute ϕ, the best AAMC obtained so far. Note that the exact AAMC $\phi = \Psi(\mathbf{Y})$ in Eq. (8) is hard to evaluate since $\mathbf{S}$ in Eq. (2) is the sum of an infinite series. Instead, ApproxAAMC performs a finite number $t = \frac{1}{\alpha}$ of iterations in Eq. (2) to obtain an approximate AAMC, since the expected length of an attributed random walk is $\frac{1}{\alpha}$. Specifically, given $\mathbf{P}_V, \widehat{\mathbf{R}}, \mathbf{R}, \alpha, \beta$ and $\mathbf{Y}$ as inputs, AppoxAAMC first initializes $\mathbf{H}_0$ as $(\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}$ and the number of iterations t to $\frac{1}{\alpha}$ (Line 1 of Algorithm 4). Then, it computes the intermediate result $\mathbf{H}_t$ by t iterations in Lines 2-3. Lastly AppoxAAMC computes ϕ with $\mathbf{H}_t$ and $\mathbf{H}_0$ at Line 4. Algorithm 4 takes $O(\frac{k}{\alpha}\cdot(|E_V| + |E_R|))$ time with the precomputed $\widehat{\mathbf{R}}$.

Utilizing algorithms GenNCI and AppoxAAMC, ACMin obtains the binary NCI $\mathbf{Y}_\ell$ and its corresponding quality measure ϕ_ℓ for each iteration ℓ, after obtaining $\mathbf{F}_\ell$. ACMin may terminate upon convergence, or reaching a preset maximum number of iterations t_e. In our experiments, we found that $t_e = 200$ is usually sufficiently large for convergence.

Next we analyze the total time and space complexities of ACMin . The computation of $\widehat{\mathbf{R}}$ at Line 1 in Algorithm 1 takes $O(|E_R|)$ time. Algorithm 4 requires $O(nk + \frac{k}{\alpha}\cdot(|E_V| + |E_R|))$ time. In each iteration (Lines 6-12), Line 7 takes $O(k\cdot(|E_V| + |E_R|))$ time and the QR decomposition over $\mathbf{Z}_\ell$ takes $O(nk^2)$ time. According to Section 4.2 and Section 4.3, GenNCI and InitNCI run in $O(t_m\cdot(nk^2 + k^3))$ and $O(\frac{k}{\alpha}\cdot|E_V|)$ time, respectively. Thus, the total time complexity of ACMin is $O\left(k(\frac{1}{\alpha} + t_e)\cdot(|E_V| + |E_R|) + nk^2 t_e t_m + \frac{kt_e}{\alpha}\cdot|E_V|)\right)$ when $k \ll n$, which equals $O(|E_V| + |E_R|)$ when t_e, t_m and k are regarded as constants. The space overhead incurred by ACMin is determined by the storage of $\mathbf{P}_V, \widehat{\mathbf{R}}, \mathbf{R}, \mathbf{Z}_\ell, \mathbf{F}_\ell$ and $\mathbf{H}_\ell$, which is bounded by $O(|E_V| + |E_R| + nk)$.

Table 2: Datasets. (K=10^3, M=10^6, B=10^9)

Name	$\lvert V\rvert$	$\lvert E_V\rvert$	$\lvert R\rvert$	$\lvert E_R\rvert$	$\lvert C\rvert$
Cora [29, 52, 53, 60, 64]	2.7K	5.4K	1.4K	49.2K	7
Citeseer [29, 52, 53, 60, 64]	3.3K	4.7K	3.7K	105.2K	6
Pubmed [52, 60, 64, 66]	19.7K	44.3K	0.5K	988K	3
Flickr [24, 29, 34, 58, 60]	7.6K	479.5K	12.1K	182.5K	9
TWeibo [60]	2.3M	50.7M	1.7K	16.8M	8
MAG-Scholar-C [3]	10.5M	265.2M	2.78M	1.1B	8

5 EXPERIMENTS

We experimentally evaluate ACMin against 11 competitors in terms of both clustering quality and efficiency on 6 real-world datasets. All experiments are conducted on a Linux machine powered by an Intel Xeon(R) Gold 6240@2.60GHz CPU and 377GB RAM. Source codes of all competitors are obtained from the respective authors.

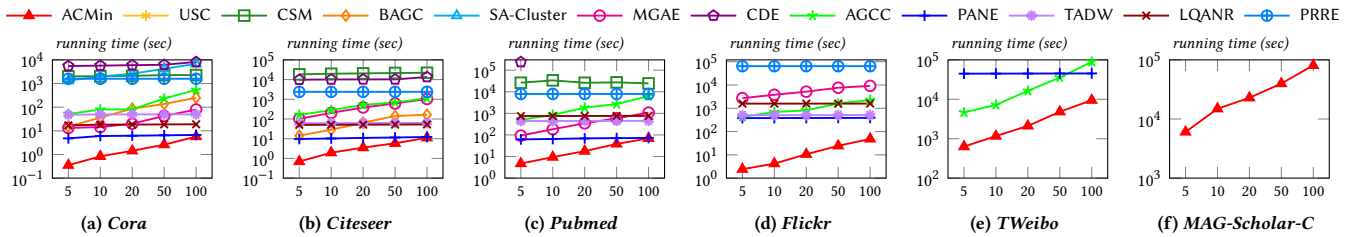
5.1 Experimental Setup

Datasets. Table 2 shows the statistics of the 6 real-world directed attributed graphs used in our experiments. $|V|$ and $|E_V|$ denote the number of nodes and edges, while $|R|$ and $|E_R|$ represent the number of attributes and node-attribute associations, respectively. $|C|$ is the number of ground-truth clusters in G. In particular, *Cora*[3], *Citeseer*[3] *Pubmed*[3] and *MAG-Scholar-C*[4] are citation graphs, in which each node represents a paper and each edge denotes a citation relationship. *Flickr*[5] and *TWeibo*[6] are social networks, in which each node represents a user, and each directed edge represents a following relationship. Further, notice that all 6 datasets have ground-truth cluster labels, and the number of ground-truth clusters $|C|$ is also included in Table 2.

Competitors. We compare ACMin with 11 competitors, including 7 k-AGC algorithms (CSM [36], SA-Cluster [67], BAGC [54], MGAE [53], CDE [29], AGCC [64], USC [50]), and 4 recent attributed network embedding algorithms (TADW [55], PANE [60], LQANR [56], PRRE [66]). The network embedding competitors are used together with k-Means to produce clustering results. In addition, we also compare with the classic unnormalized spectral clustering method USC [50], which directly works on $\mathbf{S}$ to extract clusters by materializing $\mathbf{S}$, computing the top-k eigenvectors of $\mathbf{S}$, and then applying k-Means on the eigenvectors.

Parameter settings. We adopt the default parameter settings of all competitors as suggested in their corresponding papers.

[3] http://linqs.soe.ucsc.edu/data (accessed October, 2020)
[4] https://figshare.com/articles/dataset/mag_scholar/12696653 (accessed October, 2020)
[5] https://github.com/xhuang31/LANE (accessed October, 2020)
[6] https://www.kaggle.com/c/kddcup2012-track1 (accessed October, 2020)



(a) *Cora* (b) *Citeseer* (c) *Pubmed* (d) *Flickr* (e) *TWeibo* (f) *MAG-Scholar-C*

Figure 2: Running time with varying k (best viewed in color).

Table 3: CA, NMI and AAMC with ground-truth (Large CA, NMI, and small AAMC indicate high clustering quality).

Solution	*Cora*			*Citeseer*			*Pubmed*			*Flickr*			*TWeibo*			*MAG-Scholar-C*		
	CA	NMI	AAMC	CA	NMI	AAMC	CA	NMI	AAMC	CA	NMI	AAMC	CA	NMI	AAMC	CA	NMI	AAMC
Ground-truth	1.0	1.0	0.546	1.0	1.0	0.531	1.0	1.0	0.505	1.0	1.0	0.691	1.0	1.0	0.719	1.0	1.0	0.63
TADW	0.554	0.402	0.593	0.539	0.333	0.569	0.483	0.096	0.55	0.16	0.062	0.733	-	-	-	-	-	-
LQANR	0.64	0.492	0.559	0.587	0.374	0.549	0.403	0.022	0.612	0.127	0.002	0.739	-	-	-	-	-	-
PRRE	0.547	0.396	0.604	0.576	0.322	0.592	0.62	0.269	0.518	0.454	0.321	0.713	-	-	-	-	-	-
PANE	0.601	0.462	0.577	0.677	0.421	0.537	0.618	0.252	0.512	0.402	0.265	0.708	0.215	0.004	0.752	-	-	-
CSM	0.308	0.149	0.612	0.247	0.11	0.615	0.393	0.022	0.565	-	-	-	-	-	-	-	-	-
SA-Cluster	0.001	0.01	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
BAGC	0.001	0.134	-	0.183	0	-	-	-	-	-	-	-	-	-	-	-	-	-
MGAE	0.633	0.456	0.571	0.661	0.408	0.545	0.419	0.076	0.556	0.266	0.109	0.729	-	-	-	-	-	-
CDE	0.473	0.332	0.581	0.535	0.318	0.571	0.663	0.259	0.547	0.254	0.11	0.714	-	-	-	-	-	-
AGCC	0.642	0.496	0.553	0.668	0.409	0.526	0.668	0.272	0.492	0.471	0.369	0.706	0.406	0.007	0.723	-	-	-
USC	0.635	0.455	0.706	0.495	0.326	0.682	0.548	0.212	0.614	-	-	-	-	-	-	-	-	-
ACMin	**0.656**	**0.498**	**0.544**	**0.68**	**0.422**	**0.525**	**0.691**	**0.308**	**0.487**	**0.757**	**0.608**	**0.698**	**0.408**	**0.01**	**0.686**	**0.659**	**0.497**	**0.57**

Specifically, for attributed network embedding competitors, we set the embedding dimensionality to 128. For ACMin , we set $t_e = 200, t_m = 50, \alpha = 0.2$, and $\beta = 0.35$. Competitor USC shares the same parameter settings of α, β, and t_e with ACMin .

Evaluation criteria. For efficiency evaluation, we vary the number of clusters k in $\{5, 10, 20, 50, 100\}$, and report the running time (seconds) of each method on each dataset in Section 5.2. The reported running time does not include the time for loading datasets. We terminate a method if it fails to return results within 3 days. In terms of clustering quality, we report the proposed *AAMC* measure (*i.e.*, average attributed multi-hop conductance), *modularity* [38], *CA* (clustering accuracy with respect to ground truth labels) and *NMI* (normalized mutual information) [1] to measure the clustering quality in Section 5.3. Note that AAMC considers both graph topology and node attributes to measure clustering quality, while modularity only considers graph topology. Also, note that CA and NMI rely on ground-truth clusters, while AAMC and modularity do not. Therefore, when evaluating by CA and NMI, we set k to be $|C|$ as in Table 2 for each dataset; when evaluating by modularity and AAMC, we vary k in $\{5, 10, 20, 50, 100\}$.

5.2 Efficiency Evaluation

Figure 2 presents the running time of all methods on all datasets when varying the number of clusters k in $\{5, 10, 20, 50, 100\}$. The y-axis is the running time (seconds) in log-scale. As shown in Figure 2, ACMin is consistently faster than all competitors on all datasets, often by up to orders of magnitude. ACMin is highly efficient on large attributed graphs, *e.g.*, *TWeibo* and *MAG-Scholar-C* in Figures 2e and 2f, while most of the 11 competitors fail to return results within three days. For instance, in Figure 2e, when $k = 5$, ACMin needs 630 seconds to finish, which is 7.4× faster than AGCC (4634 seconds) and 71× faster than PANE (44658 seconds), respectively. Further, ACMin is the only method able to finish on *MAG-Scholar-C* dataset that has 265.2 million edges and 1.1 billion attribute values. Specifically, ACMin only needs 1.68 hours when $k = 5$. The high efficiency of ACMin on massive real datasets is due to the its high scalable algorithmic components, whose total cost is linear to the size of the input graph as analyzed in Section 4.4. On small/moderate-sized attributed graphs in Figures 2a-2d, ACMin is also significantly faster than the competitors, especially when k is small. For instance, when $k = 10$, on *Flickr* in Figure 2d, ACMin takes 4 seconds, while the fastest competitor PANE needs 381 seconds. Note that the total running time of ACMin increases linearly with k, which is consistent with our time complexity analysis in Section 4.4 when the number of edges $|E_V| + |E_R|$ far exceeds the number of nodes n. The running time results for the 4 attributed network embedding competitors (*i.e.*, TADW, LQANR, PRRE, and PANE) are not sensitive to k, since their cost is dominated by the node embedding computation rather than k-Means . Even on settings with a large k, ACMin is still faster than all these competitors.

5.3 Quality Evaluation

CA, NMI and AAMC with ground-truth. Table 3 reports the CA, NMI, and AAMC scores of all methods, comparing to the ground-truth clusters of each dataset in Table 2. We also report the AAMC values of the ground-truth labels, which are the lower than the AAMC obtained by all methods except for ACMin , which explicitly aims to minimize AAMC. Meanwhile, we observe that relative performance of all methods measured by AAMC generally agrees with

CA. These results demonstrate that AAMC effectively measures effectively reflects clustering quality.

ACMin clearly and consistently achieves the best CA, NMI, and AAMC on all datasets. Specifically, on small attributed graphs, *i.e.*, *Cora*, *Citeseer* and *Pubmed*, compared with the best competitors (underlined in Table 3) ACMin improves CA by 1.4%, 0.3%, and 2.3%, and NMI by 0.2%, 0.1%, and 3.6%, respectively. The CA and NMI of ACMin are also significantly better than the competitors on moderate-sized/large attributed graphs (*i.e.*, *Flickr*, *TWeibo*, and *MAG-Scholar-C*). For instance, on *Flickr*, ACMin has CA 75.7%, which is 28.6% higher than that of the best competitor AGCC, which is only 47.1%. On *TWeibo*, ACMin is slightly better than AGCC; note that on this dataset, ACMin is orders of magnitude faster than AGCC as shown in Figure 2e. Hence, ACMin is overall preferable than AGCC in these settings. Finally, ACMin is the only k-AGC method capable of handling *MAG-Scholar-C*, and achieves CA 65.9% and NMI 49.7%. The superior clustering quality achieved by ACMin demonstrates the effectiveness of the proposed AAMC optimization objective in Section 2, as well as our approximate solution for this optimization program described in Section 4.

AAMC with varying k. Figure 3 reports the AAMC achieved by ACMin against all competitors on all datasets when varying the number k of clusters in $\{5, 10, 20, 50, 100\}$. Observe that ACMin consistently produces the smallest AAMC under all k settings on all datasets (smaller AAMC indicates better results), which confirms that the proposed ACMin algorithm (Algorithm 1) effectively minimizes the proposed AAMC objective function defined in Section 2.4. In particular, as shown in Figure 3, when $k = 10$, ACMin has AAMC better than the best competitor by a margin of 0.84%, 0.36%, 0.71%, 0.84% and 2.6% on *Cora*, *Citeseer*, *Pubmed*, *Flickr* and *TWeibo* respectively. Figure 3f reports the AAMC achieved by ACMin on MAG-Scholar-C, which is the only method able to return results. Further, considering the relative performance of all methods measured by CA and NMI generally agree with that measured by AAMC as shown in the results in Table 3, and the fact that ACMin is far more efficient and scalable compared to its competitors as shown in Section 5.2, we conclude that ACMin is the method of choice for k-AGC on massive graphs in practice.

Modularity with varying k. Figure 4 reports the *modularity* of all methods on all datasets when varying k in $\{5, 10, 20, 50, 100\}$. Again, observe that, for all settings of k and all datasets (except *TWeibo*), ACMin has the highest modularity. In particular, ACMin obtains a substantial imporvement of up to 5%, 4.7%, 3.8%, and 4.1% on *Cora*, *Citeseer*, *Pubmed* and *Flickr*, compared to the best competitor, respectively. Note that modularity only considers graph topology and ignores node attributes, indicating that modularity may not be able to fully evaluate clustering quality of *attributed* graphs. This may explain why on *TWeibo* the modularity of ACMin is slightly lower than some competitors. Even so, ACMin still achieves high modularity under most cases, meaning that the proposed attributed random walk model can still preserve graph topological features for clustering, in addition to node attributes.

5.4 Convergence Analysis of ACMin

In this section, we evaluate the convergence properties of ACMin , focusing on the effects of the greedy initialization technique InitNCI described in Section 4.3 on convergence speed. In particular, we compare ACMin with an ablated version ACMin-RI that replaces InitNCI at Line 2 of Algorithm 1 with random initialization of $\mathbf{Y}_0$. The number k of clusters to be detected is set to be $|C|$ as in Table 2 for each dataset. Figure 5 reports the AAMC (*i.e.*, $\Psi(\mathbf{Y})$ in Eq. (8)) produced by ACMin and ACMin-RI per iterations (Lines 6-12 in Algorithm 1), when t_e is set to 200. Observe that the AAMC produced by ACMin decreases significantly faster than that of ACMin-RI in the early iterations, and also converges faster than ACMin-RI. For instance, in Figure 5b, on *Citeseer*, ACMin requires about 80 iterations to reach a plateaued AAMC, while ACMin-RI needs 140 iterations. Moreover, GenNCI is able to help ACMin to achieve lower AAMC at convergence as shown in Figure 5. This experimental evaluation demonstrates the efficiency and effectiveness of the proposed greedy initialization technique in Section 4.3.

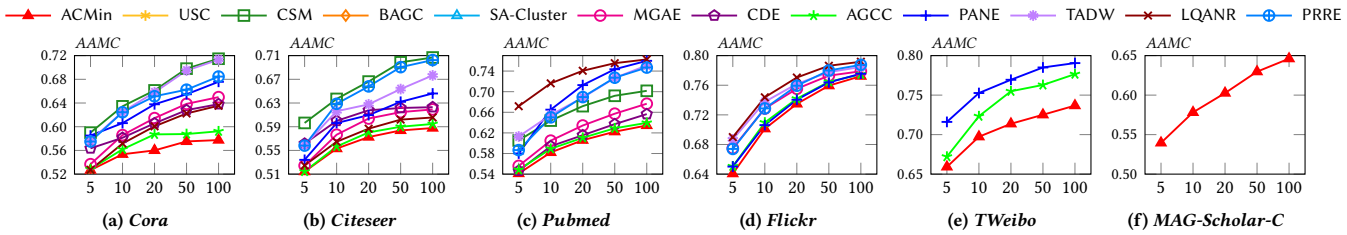
6 RELATED WORK

Attributed graph clustering has been extensively studied in literature, as surveyed in [4, 5, 11]. In the following, we review the existing methods that are most relevant to this work.

Edge-weight-based clustering. A classic methodology is to convert the input attributed graph to a weighted graph by assigning each edge a weight based on the attribute and topological similarity between the two nodes of the edge; then, traditional weighted graph clustering algorithms are directly applied [7, 33, 37, 42, 45]. For instance, Neville *et al.* [37] assign a weight to each edge (u, v) of the input attributed graph G based on the number of attribute values that u and v have in common, and construct a weighted graph G'. Then they apply the classic spectral clustering [50] over G' to produce clusters. However, these methods only consider the attributes of two directly connected nodes and use hand-crafted weights to represent attributes, and thus, result in inferior clustering quality.

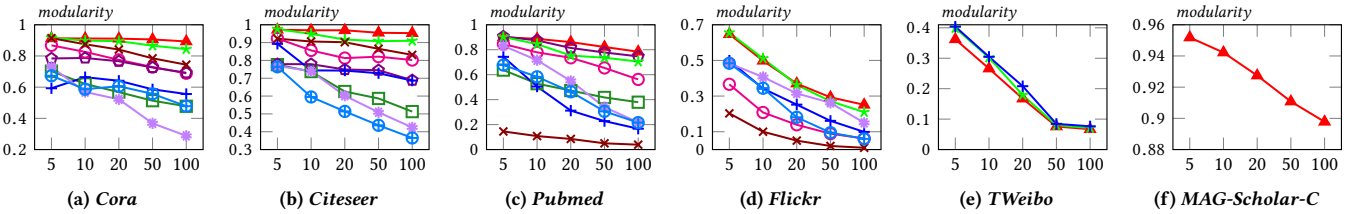
Distance-based clustering. Existing distance-based clustering solutions construct a distance matrix $\mathbf{M}$ by combining the topological and attribute similarity between nodes, and then apply classic distance-based clustering methods, such as k-Means [19] and k-Medoids [41], on $\mathbf{M}$ to generate clusters. For instance, SA-Cluster [67] extends the original input attributed graph G to an attribute-augmented graph G' by treating each attribute as a node, and then samples random walks over G' to compute the distance between nodes in G', in order to construct $\mathbf{M}$, which is then fed into a k-Centroids method to generate clusters. Further, DCom [7] applies hierarchical agglomerative clustering on a constructed distance matrix. CSM [36] computes the distance matrix $\mathbf{M}$ based on a shortest path strategy that considers both structural and attribute relevance among nodes, and applies k-Medoids over $\mathbf{M}$ to generate clusters. ANCA [10] applies k-Means for the sum of eigenvectors of the distance and similarity matrices to generate clusters. Distance-based clustering methods suffer from severe efficiency issues since they require to compute the distance of every node pair, resulting in $O(n^2)$ time and space overhead, which is prohibitive in practice. For instance, as shown in our experiments, both SA-Cluster and CSM suffer from costly running time and poor clustering quality.

Probabilistic-model-based clustering. Based on the assumption that the structure, attributes, and clusters of attributed graphs are



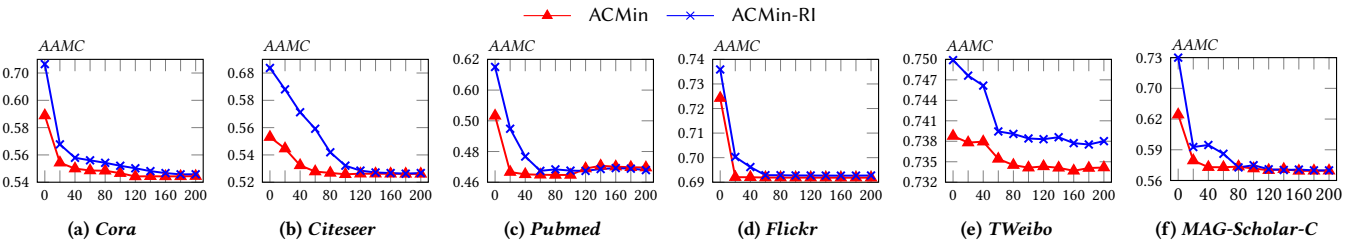
(a) *Cora* (b) *Citeseer* (c) *Pubmed* (d) *Flickr* (e) *TWeibo* (f) *MAG-Scholar-C*

Figure 3: AAMC with varying k (best viewed in color).



(a) *Cora* (b) *Citeseer* (c) *Pubmed* (d) *Flickr* (e) *TWeibo* (f) *MAG-Scholar-C*

Figure 4: Modularity with varying k (best viewed in color).



(a) *Cora* (b) *Citeseer* (c) *Pubmed* (d) *Flickr* (e) *TWeibo* (f) *MAG-Scholar-C*

Figure 5: AAMC with varying t_e (best viewed in color).

generated according to a certain parametric distribution, there exist a collection of probabilistic-model-based clustering methods, which statistically infer a probabilistic model for attributed graph clustering, in order to generate clustering results. In particular, PCL-DC [62] combines a conditional model of node popularity and a discriminative model that reduces the impact of irrelevant attributes into a unified model, and then finds the clustering result that optimizes the model. CohsMix [63] formulates the clustering problem by MixNet model [40] and then utilizes a varient of EM algorithm to optimize it, in order to generate clustering results. BAGC [54] designs a generative Bayesian model [8] that produces a sample of all the possible combinations of a graph based on adjacency matrix $\mathbf{A}$ and attribute matrix $\mathbf{X}$, and aims to find a clustering result C maximizing a conjoint probability $\mathbb{P}(C|\mathbf{A},\mathbf{X})$. Note that the optimization process to estimate the likelihood parameters in these probabilistic-model-based clustering methods often incurs substantial time overheads, as validated in our experiments (Section 5.2).

Embedding-based methods. In recent years, a plethora of network embedding techniques are proposed for attributed graphs. The objective of network embedding is to learn an embedding vector for each node such that the graph topology and attribute information surrounding the nodes can be preserved. We can directly employ traditional clustering methods (*e.g.*, k-Means) over the embedding vectors to generate clusters [19, 41]. AA-Cluster [2] builds a weighted graph based on graph topology and node attributes, and then applies network embedding on the weighted graph to generate embeddings. MGAE [53] proposes a marginalized graph convolutional network to learn embeddings. CDE [29] learns node embeddings by optimizing a non-negative matrix factorization problem based on community structure embeddings and node attributes. DAEGC [52] fuses graph topology and node attributes via an attention-based autoencoder [48] to obtain embeddings, and then generates soft labels to guide a self-training graph clustering procedure. AGCC[64] utilizes an adaptive graph convolution method to learn embeddings, and then applies the spectral clustering on the similarity matrix computed from the learnt embeddings to obtain clusters. The above methods either incur immense overheads in learning embeddings or suffer from unsatisfactory clustering quality. There are many attributed network embedding methods proposed, *e.g.*, [17, 31, 34, 55–57, 60, 66]. However, most of them are not specially designed for clustering purpose, leading to suboptimal clustering quality, as demonstrated in our experiments when comparing with TADW, LQANR, PRRE and PANE.

7 CONCLUSIONS

This paper presents ACMin , an effective and scalable solution for k-AGC computation. ACMin achieves high scalability and effectiveness through a novel problem formulation based on the proposed attributed multi-hop conductance measure for cluster quality, as well as a carefully designed iterative optimization framework and an effective greedy clustering initialization method. Extensive experiments demonstrate that ACMin achieves substantial performance gains over the previous state of the art in terms of both efficiency and clustering quality. Regarding future work, we plan to study parallelized versions of ACMin , running on multi-core CPUs and GPUs, as well as in a distributed setting with multiple servers, in order to handle even larger datasets. Meanwhile, we intend to extend ACMin to handle attributed heterogeneous graphs with different types of nodes and edges.

ACKNOWLEDGMENTS

This work is supported by the National University of Singapore under SUG grant R-252-000-686-133, by NPRP grant #NPRP10-0208-170408 from the Qatar National Research Fund (a member of Qatar Foundation). Jieming Shi is supported by the financial support(1-BE3T) of research project (P0033898) from Hong Kong Polytechnic University. The findings herein reflect the work, and are solely the responsibility, of the authors.

A PROOFS

Proof of Lemma 2.3. Let $p_\ell(v_i, v_j)$ be the probability that an attributed random walk starting from v_i stops at v_j at the ℓ-th hop. We first prove that

$$p_\ell(v_i, v_j) = \alpha(1-\alpha)^\ell \cdot ((1-\beta)\cdot \mathbf{P}_V + \beta \cdot \mathbf{P}_R)^\ell[v_i, v_j]. \quad (18)$$

Note that if Eq. (18) holds, the overall probability that an attributed random walk from v_i terminates at v_j is $\sum_{\ell=0}^{\infty} p_\ell(v_i, v_j) = \mathbf{S}[v_i, v_j]$, which establishes the equivalence in Eq. (2). To this end, we prove Eq. (18) by induction. First, let us consider the initial case that the attributed random walk terminates at source node v_i with probability α. In this case, $p_0(v_i, v_j) = \alpha$ if $v_i = v_j$; otherwise $p_0(v_i, v_j) = 0$, which is identical to the r.h.s of Eq. (18) when $\ell = 0$. Therefore, Eq. (18) holds when $\ell = 0$. Assume that Eq. (18) holds at the ℓ'-th hop. Then the probability that an attributed random walk from v_i visits any node $v_l \in V$ at the ℓ'-th hop is $(1-\alpha)^{\ell'} \cdot ((1-\beta)\cdot \mathbf{P}_V + \beta \cdot \mathbf{P}_R)^{\ell'}[v_i, v_l]$. Based on this assumption, for the case $\ell = \ell' + 1$, with probability $1-\alpha$, it will navigate to node v_j according to the probability $(1-\beta)\cdot \mathbf{P}_V[v_l, v_j] + \beta \cdot \mathbf{P}_R[v_l, v_j]$, and finally stop at v_j with probability α. Thus, $p_{\ell'+1}(v_i, v_j) = \sum_{v_l \in V} (1-\alpha)^{\ell'} \cdot ((1-\beta)\cdot \mathbf{P}_V + \beta \cdot \mathbf{P}_R)^{\ell'}[v_i, v_l] \cdot (1-\alpha)\alpha((1-\beta)\cdot \mathbf{P}_V[v_l, v_j] + \beta \cdot \mathbf{P}_R[v_l, v_j]) = \alpha(1-\alpha)^{\ell'+1} \cdot ((1-\beta)\cdot \mathbf{P}_V + \beta \cdot \mathbf{P}_R)^{\ell'+1}[v_i, v_j]$, which completes the proof. □

Proof of Lemma 3.1. By Eq. (6), for cluster C_i, we have vector $((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})[c_i]$, where each entry $((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})[c_i, v_j] = 1/\sqrt{|C_i|}$ if $v_j \in C_i$ and otherwise $((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})[c_i, v_j] = 0$. Note that

$$\begin{aligned}
&2 \cdot ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})[c_i] \cdot (\mathbf{I}-\mathbf{S}) \cdot ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})[c_i]^\top \\
&= \textstyle\sum_{v_j, v_l \in V} \mathbf{S}[v_j, v_l] \cdot (((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})[c_i, v_j] - ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})[c_i, v_l])^2 \\
&= \textstyle\sum_{v_j \in C_i, v_l \in V \setminus C_i} \mathbf{S}[v_j, v_l] \cdot ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})[c_i, v_j]^2 = \frac{\Phi(C_i)}{2}.
\end{aligned}$$

Then we have

$$\begin{aligned}
\frac{\sum_{i=1}^k \Phi(C_i)}{k} &= \frac{2}{k} \textstyle\sum_{c_i=1}^k ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})[c_i] \cdot (\mathbf{I}-\mathbf{S}) \cdot ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})[c_i]^\top \\
&= \frac{2}{k} \cdot \mathrm{trace}(((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}) \cdot (\mathbf{I}-\mathbf{S}) \cdot ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})^\top),
\end{aligned}$$

which completes our proof. □

Proof of Lemma 3.2. First, we construct a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ based on the input graph $G = (V, E_V, R, E_R)$ by letting $\mathcal{V} = V$ and $\mathcal{E} = \{(v_i, v_j, \mathbf{S}[v_i, v_j]) \mid v_i, v_j \in V \text{ and } \mathbf{S}[v_i, v_j] > 0\}$, where $\mathbf{S}[v_i, v_j]$ signifies the weight of edge (v_i, v_j). Thus, Eq. (8) can be reduced to the objective function of the min-cut problem on $\mathcal{G}$, which is proven to be NP-hard in [14, 51]. □

Proof of Lemma 3.3. Let $\lambda_i(\mathbf{M})$ be the i-th smallest eigenvalue of matrix $\mathbf{M}$. Note that $\forall \mathbf{Y} \in \mathbb{R}^{k\times n}$, $((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}) \cdot ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})^\top = \mathbf{I}$, meaning that $f(\mathbf{Y}) = ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})^\top \cdot ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})$ is a projection matrix of rank k. Therefore, $\forall i < n-k+1, \lambda_i(f(\mathbf{Y})) = 0$ and $\forall i \geq n-k+1, \lambda_i(f(\mathbf{Y})) = 1$. By Von Neumann's trace inequality [35] and the property of matrix trace, for any $\mathbf{Y} \in \mathbb{R}^{k\times n}$, we have the following inequality:

$$\begin{aligned}
\Psi(\mathbf{Y}) &= \frac{2}{k} \cdot \mathrm{trace}(((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}) \cdot (\mathbf{I}-\mathbf{S}) \cdot ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})^\top) \\
&= \frac{2}{k} \cdot \mathrm{trace}((\mathbf{I}-\mathbf{S}) \cdot f(\mathbf{Y})) \geq \frac{2}{k} \cdot \textstyle\sum_{i=1}^n \lambda_i(\mathbf{I}-\mathbf{S}) \cdot \lambda_{n-i+1}(f(\mathbf{Y})) \\
&= \frac{2}{k} \cdot \textstyle\sum_{i=1}^k \lambda_i(\mathbf{I}-\mathbf{S}) = \frac{2}{k} \cdot \sum_{i=1}^k (1-\lambda_{n-i+1}(\mathbf{S})). \quad (19)
\end{aligned}$$

Note that $\mathbf{F}$ be the top-k eigenvectors of $\mathbf{S}$, implying $\mathbf{F}\mathbf{F}^\top = \mathbf{I}$ and $(\mathbf{I}-\mathbf{S}) \cdot \mathbf{F}[c_i]^\top = (1-\lambda_{n-i+1}(\mathbf{S})) \cdot \mathbf{F}[c_i]^\top$ for $1 \leq i \leq k$. Hence,

$$\frac{2}{k} \cdot \mathrm{trace}(\mathbf{F}(\mathbf{I}-\mathbf{S})\mathbf{F}^\top) = \frac{2}{k} \cdot \textstyle\sum_{i=1}^k (1-\lambda_{n-i+1}(\mathbf{S})), \quad (20)$$

which implies that $\Psi(\mathbf{Y})$ is minimized when $((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}) = \mathbf{F}$. Suppose $\mathbf{Y}^* \in \mathbb{R}^{k\times n}$ is the optimal solution to Eq. (8). Therefore, with Eq. (19) and Eq. (20), the following inequality holds

$$\begin{aligned}
\phi^* = \Psi(\mathbf{Y}^*) &= \frac{2}{k} \cdot \mathrm{trace}(((\mathbf{Y}^*\mathbf{Y}^{*\top})^{-\frac{1}{2}}\mathbf{Y}^*)(\mathbf{I}-\mathbf{S})((\mathbf{Y}^*\mathbf{Y}^{*\top})^{-\frac{1}{2}}\mathbf{Y}^*)^\top) \\
&\geq \frac{2}{k} \cdot \textstyle\sum_{i=1}^k (1-\lambda_{n-i+1}(\mathbf{S})) = \frac{2}{k} \cdot \mathrm{trace}(\mathbf{F}(\mathbf{I}-\mathbf{S})\mathbf{F}^\top),
\end{aligned}$$

which finishes our proof. □

Proof of Lemma 3.4. Eq. (9) implies that $\mathbf{X}\mathbf{F} \to (\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}$ where $\mathbf{X}^\top\mathbf{X} = \mathbf{I}$. By the property of matrix trace, we have

$$\begin{aligned}
\Psi(\mathbf{Y}) &= \frac{2}{k} \cdot \mathrm{trace}((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y} \cdot (\mathbf{I}-\mathbf{S}) \cdot ((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y})^\top) \\
&\to \frac{2}{k} \cdot \mathrm{trace}(\mathbf{X}\mathbf{F}(\mathbf{I}-\mathbf{S})\mathbf{F}^\top\mathbf{X}^\top) \\
&= \frac{2}{k} \cdot \mathrm{trace}(\mathbf{X}^\top\mathbf{X}\mathbf{F}(\mathbf{I}-\mathbf{S})\mathbf{F}^\top) = \frac{2}{k} \cdot \mathrm{trace}(\mathbf{F}(\mathbf{I}-\mathbf{S})\mathbf{F}^\top). \quad (21)
\end{aligned}$$

By Lemma 3.3, we have $\Psi(\mathbf{Y}) \to \phi^*$, completing our proof. □

Proof of Lemma 4.1. We need the following lemmas for the proof.

LEMMA A.1 ([65]). *If $[\lambda, \mathbf{x}]$ is an eigen-pair of matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$, then $[\sum_{\ell=0}^t w_\ell \lambda^\ell, \mathbf{x}]$ is an eigen-pair of matrix $\sum_{\ell=0}^t w_\ell \mathbf{M}^\ell$.*

Lemma A.2 ([20]). *Given a* $\mathbf{M} \in \mathbb{R}^{n\times n}$ *satisfying* $\sum_{j=1}^{n} \mathbf{M}[i,j] = 1\ \forall 1 \le i \le n$ *and each entry* $\mathbf{M}[i,j] \ge 0\ \forall 1 \le i,j \le n$, *the largest eigenvalue* λ_1 *of* $\mathbf{M}$ *is 1.*

Suppose that $[\lambda_i, \mathbf{x}_i]$ is an eigen-pair of $(1-\beta)\cdot\mathbf{P}_V + \beta\cdot\mathbf{P}_R$ and λ_i is its i-th largest eigenvalue. Note that each row sum of $(1-\beta)\cdot\mathbf{P}_V + \beta\cdot\mathbf{P}_R$ is equal to 1 and each entry of $(1-\beta)\cdot\mathbf{P}_V + \beta\cdot\mathbf{P}_R$ is non-negative. Then, by Lemma A.2, we have $\lambda_i \in [-1,1]$ for $1 \le i \le n$. Let $f(\lambda_i) = \sum_{\ell=0}^{t} \alpha(1-\alpha)^\ell \lambda_i^\ell$. Lemma A.1 implies that any eigen-pair $\forall i \in [1,n], [f(\lambda_i), \mathbf{x}_i]$ of $(1-\beta)\cdot\mathbf{P}_V + \beta\cdot\mathbf{P}_R$ is an eigen-pair of $\mathbf{S}$. By the sum of geometric sequence, we have $f(\lambda_i) = \alpha\cdot\frac{1-(1-\alpha)^{t+1}\lambda_i^{t+1}}{1-(1-\alpha)\lambda_i} = \frac{\alpha}{1-(1-\alpha)\lambda_i}$, which is is monotonously decreasing when $1 \le i \le n$. Hence, for $1 \le i \le n$, $f(\lambda_i)$ and $\mathbf{x}_i$ are the i-th largest eigenvalue and the i-th largest eigenvector of $\mathbf{S}$. Recall that $\mathbf{F}$ is the top-k eigenvectors of $(1-\beta)\cdot\mathbf{P}_V + \beta\cdot\mathbf{P}_R$. Therefore, $\mathbf{F}$ is the top-k eigenvectors of $\mathbf{S}$. The lemma is proved. □

Proof of Lemma 4.2. Let $\mathbf{Z} = \mathbf{V}^\top\mathbf{X}^\top\mathbf{U}$. Since $\mathbf{U}$ and $\mathbf{V}$ are the left and right singular vectors, we have $\mathbf{U}\mathbf{U}^\top = \mathbf{I}$ and $\mathbf{V}\mathbf{V}^\top = \mathbf{I}$. Note that $\mathbf{Z}\mathbf{Z}^\top = \mathbf{I}$, which implies that each $\mathbf{Z}[i,j]$ satisfies $-1 \le \mathbf{Z}[i,j] \le 1$. Also, $\Sigma[i,i]$ is a singular value and thus $\Sigma[i,i] > 0$. Then,

$$\begin{aligned}\text{trace}((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}\mathbf{F}^\top\mathbf{X}^\top) &= \text{trace}(\mathbf{U}\Sigma\mathbf{V}^\top\mathbf{X}^\top) = \text{trace}(\Sigma\mathbf{V}^\top\mathbf{X}^\top\mathbf{U})\\ &= \sum_{i=1}^{k}\Sigma[i,i]\cdot\mathbf{Z}[i,i] \le \sum_{i=1}^{k}\Sigma[i,i].\end{aligned}$$

Therefore, $\text{trace}((\mathbf{Y}\mathbf{Y}^\top)^{-\frac{1}{2}}\mathbf{Y}\mathbf{F}^\top\mathbf{X}^\top)$ is maximized when $\mathbf{Z} = \mathbf{I}$, which implies that $\mathbf{X} = \mathbf{U}\mathbf{V}^\top$. The lemma is proved. □

REFERENCES

[1] Charu C Aggarwal and Chandan K Reddy. 2014. *Data Clustering: Algorithms and Applications.* CRC Press.
[2] Esra Akbas and Peixiang Zhao. 2017. Attributed graph clustering: An attribute-aware graph embedding approach. In *ASONAM.*
[3] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling Graph Neural Networks with Approximate PageRank. In *SIGKDD.*
[4] Cécile Bothorel, Juan David Cruz, Matteo Magnani, and Barbora Micenkova. 2015. Clustering attributed graphs: models, measures and methods. *Network Science* (2015).
[5] Petr Chunaev. 2019. Community detection in node-attributed social networks: a survey. *arXiv preprint arXiv:1912.09816* (2019).
[6] Fan RK Chung and Fan Chung Graham. 1997. *Spectral graph theory.*
[7] David Combe, Christine Largeron, Elöd Egyed-Zsigmond, and Mathias Géry. 2012. Combining relations and text in scientific network clustering. In *ASONAM.*
[8] Peter Congdon. 2007. *Bayesian statistical modelling.*
[9] James W Demmel. 1997. *Applied numerical linear algebra.* Siam.
[10] Issam Falih, Nistor Grozavu, Rushed Kanawati, and Younès Bennani. 2017. Anca: Attributed network clustering algorithm. In *Complex Networks.*
[11] Issam Falih, Nistor Grozavu, Rushed Kanawati, and Younès Bennani. 2018. Community detection in attributed network. In *WWW.*
[12] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* (2010).
[13] Linton C Freeman. 1996. Cliques, Galois lattices, and the structure of human social groups. *Social networks* (1996).
[14] Olivier Goldschmidt and Dorit S Hochbaum. 1988. Polynomial algorithm for the k-cut problem. In *FOCS.*
[15] Roger Guimera and Luis A Nunes Amaral. 2005. Functional cartography of complex metabolic networks. *Nature* (2005).
[16] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* (2011).
[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS.*
[18] Daniel Hanisch, Alexander Zien, Ralf Zimmer, and Thomas Lengauer. 2002. Co-clustering of biological networks and gene expression data. *Bioinformatics* (2002).
[19] John A Hartigan and Manchek A Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *J R Stat Soc Ser C* (1979).
[20] Taher Haveliwala and Sepandar Kamvar. 2003. *The second eigenvalue of the Google matrix.* Technical Report. Stanford.
[21] Dongxiao He, Zhiyong Feng, Di Jin, Xiaobao Wang, and Weixiong Zhang. 2017. Joint identification of network communities and semantics via integrative modeling of network topologies and node contents. In *AAAI.*
[22] Darko Hric, Richard K Darst, and Santo Fortunato. 2014. Community detection in networks: Structural communities versus ground truth. *Physical Review E* (2014).
[23] Huimin Huang, Hong Shen, and Zaiqiao Meng. 2020. Community-based influence maximization in attributed networks. *Applied Intelligence* (2020).
[24] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. In *WSDM.*
[25] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *WWW.*
[26] Gueorgi Kossinets and Duncan J Watts. 2006. Empirical analysis of an evolving social network. *science* (2006).
[27] Timothy La Fond and Jennifer Neville. 2010. Randomization tests for distinguishing social influence and homophily effects. In *WWW.*
[28] Andrea Lancichinetti and Santo Fortunato. 2009. Community detection algorithms: a comparative analysis. *Physical review E* (2009).
[29] Ye Li, Chaofeng Sha, Xin Huang, and Yanchun Zhang. 2018. Community detection in attributed graphs: An embedding approach. In *AAAI.*
[30] U Liji, Yahui Chai, and Jianrui Chen. 2018. Improved personalized recommendation based on user attributes clustering and score matrix filling. *CSI* (2018).
[31] Jie Liu, Zhicheng He, Lai Wei, and Yalou Huang. 2018. Content to node: Self-translation network embedding. In *SIGKDD.*
[32] László Lovász et al. 1993. Random walks on graphs: A survey. *Combinatorics, Paul erdos is eighty* (1993).
[33] Fanrong Meng, Xiaobin Rui, Zhixiao Wang, Yan Xing, and Longbing Cao. 2018. Coupled node similarity learning for community detection in attributed networks. *Entropy* (2018).
[34] Zaiqiao Meng, Shangsong Liang, Hongyan Bao, and Xiangliang Zhang. 2019. Co-embedding attributed networks. In *WSDM.*
[35] Leon Mirsky. 1975. A trace inequality of John von Neumann. *Monatshefte für mathematik* (1975).
[36] Waqas Nawaz, Kifayat-Ullah Khan, Young-Koo Lee, and Sungyoung Lee. 2015. Intra graph clustering using collaborative similarity measure. *DAPD* (2015).
[37] Jennifer Neville, Micah Adler, and David Jensen. 2003. Clustering relational data using attribute and link information. In *IJCAI.*
[38] Mark EJ Newman and Michelle Girvan. 2004. Finding and evaluating community structure in networks. *Physical review E* (2004).
[39] Andrew Y Ng, Michael I Jordan, and Yair Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *NeurIPS.*
[40] Krzysztof Nowicki and Tom A B Snijders. 2001. Estimation and prediction for stochastic blockstructures. *J Am Stat Assoc* (2001).
[41] Hae-Sang Park and Chi-Hyuck Jun. 2009. A simple and fast algorithm for K-medoids clustering. *Expert systems with applications* (2009).
[42] Yiye Ruan, David Fuhry, and Srinivasan Parthasarathy. 2013. Efficient community detection in large networks using content and links. In *WWW.*
[43] Heinz Rutishauser. 1969. Computational aspects of FL Bauer's simultaneous iteration method. *Numer. Math.* (1969).
[44] Satu Elisa Schaeffer. 2007. Graph clustering. *Computer science review* (2007).
[45] Karsten Steinhaeuser and Nitesh V Chawla. 2008. Community detection in a large real-world social network. In *SBP.*
[46] Shayan A Tabrizi, Azadeh Shakery, Masoud Asadpour, Maziar Abbasi, and Mohammad Ali Tavallaie. 2013. Personalized pagerank clustering: A graph clustering algorithm based on random walks. *Physica A* (2013).
[47] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In *ICDM.*
[48] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *ICLR* (2018).
[49] Konstantin Voevodski, Shang-Hua Teng, and Yu Xia. 2009. Finding local communities in protein networks. *BMC bioinformatics* (2009).
[50] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and computing* (2007).
[51] Dorothea Wagner and Frank Wagner. 1993. Between min cut and graph bisection. In *MFCS.*
[52] Chun Wang, Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Attributed graph clustering: a deep attentional embedding approach. In *IJCAI.*
[53] Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. 2017. Mgae: Marginalized graph autoencoder for graph clustering. In *CIKM.*
[54] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. 2012. A model-based approach to attributed graph clustering. In *SIGMOD.*
[55] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. 2015. Network representation learning with rich text information. In *AAAI.*
[56] Hong Yang, Shirui Pan, Ling Chen, Chuan Zhou, and Peng Zhang. 2019. Low-Bit Quantization for Attributed Network Representation Learning. In *IJCAI.*
[57] Hong Yang, Shirui Pan, Peng Zhang, Ling Chen, Defu Lian, and Chengqi Zhang. 2018. Binarized attributed network embedding. In *ICDM.*
[58] Jaewon Yang, Julian McAuley, and Jure Leskovec. 2013. Community detection in networks with node attributes. In *ICDM.*

[59] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S Bhowmick. 2020. Homogeneous network embedding for massive graphs via reweighted personalized PageRank. *PVLDB* (2020).
[60] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Juncheng Liu, and Sourav S. Bhowmick. 2021. Scaling Attributed Network Embedding to Massive Graphs. *PVLDB* (2021).
[61] Renchi Yang, Xiaokui Xiao, Zhewei Wei, Sourav S Bhowmick, Jun Zhao, and Rong-Hua Li. 2019. Efficient Estimation of Heat Kernel PageRank for Local Clustering. In *SIGMOD*.
[62] Tianbao Yang, Rong Jin, Yun Chi, and Shenghuo Zhu. 2009. Combining link and content for community detection: a discriminative approach. In *SIGKDD*.
[63] Hugo Zanghi, Stevenn Volant, and Christophe Ambroise. 2010. Clustering based on random graph model embedding vertex features. *Pattern Recognition Letters* (2010).
[64] Xiaotong Zhang, Han Liu, Qimai Li, and Xiao-Ming Wu. 2019. Attributed graph clustering via adaptive graph convolution. In *IJCAI*.
[65] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-order proximity preserved network embedding. In *SIGKDD*.
[66] Sheng Zhou, Hongxia Yang, Xin Wang, Jiajun Bu, Martin Ester, Pinggang Yu, Jianwei Zhang, and Can Wang. 2018. PRRE: Personalized Relation Ranking Embedding for Attributed Networks. In *CIKM*.
[67] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2009. Graph clustering based on structural/attribute similarities. *PVLDB* (2009).
[68] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2010. Clustering large attributed graphs: An efficient incremental approach. In *ICDM*.