Effective Clustering on Large Attributed Bipartite Graphs

Renchi Yang
Hong Kong Baptist University
Hong Kong SAR, China
renchi@hkbu.edu.hk

Yidu Wu*
Chinese University of Hong Kong
Hong Kong SAR, China
yiduwu@cuhk.edu.hk

Xiaoyang Lin
Hong Kong Baptist University
Hong Kong SAR, China
csxylin@hkbu.edu.hk

Qichen Wang
Hong Kong Baptist University
Hong Kong SAR, China
qcwang@hkbu.edu.hk

Tsz Nam Chan
Shenzhen University
Shenzhen, China
edisonchan@szu.edu.cn

Jieming Shi
Hong Kong Polytechnic University
Hong Kong SAR, China
jieming.shi@polyu.edu.hk

Abstract

Attributed bipartite graphs (ABGs) are an expressive data model for describing the interactions between two sets of heterogeneous nodes that are associated with rich attributes, such as customer-product purchase networks and author-paper authorship graphs. Partitioning the target node set in such graphs into k disjoint clusters (referred to as k-ABGC) finds widespread use in various domains, including social network analysis, recommendation systems, information retrieval, and bioinformatics. However, the majority of existing solutions towards k-ABGC either overlook attribute information or fail to capture bipartite graph structures accurately, engendering severely compromised result quality. The severity of these issues are accentuated in real ABGs, which often encompass millions of nodes and a sheer volume of attribute data, rendering effective k-ABGC over such graphs highly challenging.

In this paper, we propose TPO, an effective and efficient approach to k-ABGC that achieves superb clustering performance on multiple real datasets. TPO obtains high clustering quality through two major contributions: (i) a novel formulation and transformation of the k-ABGC problem based on *multi-scale attribute affinity* specialized for capturing attribute affinities between nodes with the consideration of their multi-hop connections in ABGs, and (ii) a highly efficient solver that includes a suite of carefully-crafted optimizations for sidestepping explicit affinity matrix construction and facilitating faster convergence. Extensive experiments, comparing TPO against 19 baselines over 5 real ABGs, showcase the superior clustering quality of TPO measured against ground-truth labels. Moreover, compared to the state of the arts, TPO is often more than 40× faster over both small and large ABGs.

CCS Concepts

• **Mathematics of computing** → *Computations on matrices*; • **Computing methodologies** → **Cluster analysis**; **Spectral methods**; • **Information systems** → **Clustering**.

---
*Work done while at Hong Kong Baptist University.

Keywords

clustering, bipartite graphs, attributes, eigenvector

1 Introduction

Bipartite graphs are an indispensable data structure used to model the interplay between two sets of entities from heterogeneous sources, e.g., author-publication associations, customer-merchant transactions, query-webpage pairing, and various user-item interactions on social media, e-commerce platforms, search engines, etc. In the real world, such graphs are often associated with rich attributes, e.g., the user profile in social networks, web page content in web graphs, hallmarks of pathways in cancer signaling networks, and paper keywords in academic graphs, which are termed *Attributed Bipartite Graphs* (hereinafter ABGs).

Given an ABG $\mathcal{G}$ with two disjoint node sets $\mathcal{U}$ and $\mathcal{V}$, k-*Attributed Bipartite Graph Clustering* (k-ABGC), a fundamental task of analyzing ABGs, seeks to partition the nodes in the node set of interest, e.g., $\mathcal{U}$ or $\mathcal{V}$, into k non-overlapping clusters $C_1, C_2, \cdots, C_k$, such that nodes within the same cluster C_i are close to each other in terms of both their attribute similarity and topological proximity in $\mathcal{G}$. Due to the omnipresence of ABGs, k-ABGC has seen a wide range of practical applications in social network analysis, recommender systems, information retrieval, and bioinformatics, such as user/content tagging [45, 81], market basket analysis [83, 84], document categorization [8, 59], identification of protein complexes, disease genes, and drug targets [46, 64], and many others [26, 34, 50, 67, 69].

As reviewed in Section 5, existing solutions towards k-ABGC primarily rely on *bipartite graph co-clustering* (BGCC), *attributed graph clustering* (AGC), and *attributed network embedding* (ANE) techniques. Amid them, BGCC has been extensively investigated in the literature [2, 8, 9, 28, 29, 63] for clustering non-attributed bipartite graphs, whose basic idea is to simultaneously group nodes in $\mathcal{U}$ and $\mathcal{V}$ merely based on their interactions in $\mathcal{G}$, instead of clustering them severally. As pinpointed in prior works [4], the attributes present rich information to characterize the properties of nodes and hence, can complement scant topological information for better node clustering. Consequently, BGCC methods exhibit

KDD '24, August 25–29, 2024, Barcelona, Spain

Renchi Yang et al.

subpar performance on ABGs as they overlook such information.

To leverage the complementary nature of graph topology and attributes for enhanced clustering effectiveness, considerable efforts [4, 6, 12, 30, 77, 85] have been invested in recent years towards devising effective AGC models and algorithms. Although these approaches enjoy improved performance over *unipartite* attributed graphs by fusing graph connectivity and attribute information of nodes via deep learning or sophisticated statistical models, they are sub-optimal for ABGs.

Over the past decade, network embedding has emerged as a popular and powerful tool for analyzing graph-structured data, especially those with nodal attributes. Notwithstanding a plethora of network embedding techniques invented [7, 14, 75], most of them are designed for unipartite graphs. To capture the unique characteristics of bipartite graphs, Huang et al. [23] extend node2vec [18] to ABGs, at the expense of tremendous training overhead. Adopting this category of approaches for k-ABGC requires a rear-mounted phase (e.g., k-Means) to cluster the node embeddings, which is not cost-effective given the high embedding dimensions (typically 128). To summarize, existing approaches to k-ABGC either dilute clustering quality due to inadequate exploitation of attributes and bipartite graph topology, or incur vast computation costs, especially on sizable ABGs encompassing thousands of attributes, millions of nodes, and billions of edges.

In response to these challenges, we propose TPO, a novel Three-Phase Optimization framework for k-ABGC that significantly advances the state of the art in k-ABGC, in terms of both result effectiveness and computation efficiency. First and foremost, TPO formulates the k-ABGC task as an optimization problem based on *multi-scale attribute affinity* (MSA), a new node affinity measure dedicated to ABGs. More concretely, the MSA of two homogeneous nodes u_i, u_j in $\mathcal{U}$ of ABG $\mathcal{G}$ evaluates the similarity of their attributes aggregated from multi-hop neighborhoods, which effectively captures the affinity of nodes with consideration of both their attributes and topological connections in bipartite graphs. However, calculating the MSA of all node pairs in $\mathcal{G}$ for clustering is prohibitively expensive for large graphs, as it entails colossal construction time and space consumption ($O(|\mathcal{U}|^2)$). On top of that, the exact optimization of our k-ABGC objective is also infeasible as an aftermath of its NP-hardness.

To tackle these issues, TPO adopts a three-phase optimization scheme for an approximate solution with time and space costs linear to the size of $\mathcal{G}$. Under the hood, similar in spirit to kernel tricks [36], TPO first leverages a mathematical apparatus, *random features* [49, 82], to bypass the materialization of the all-pairwise MSA. The clustering task is later framed as a non-negative matrix factorization, followed by a matrix approximation problem, based on our theoretically-grounded problem transformation. Particularly, the former attends to yielding an intermediate, while the latter iteratively refines the intermediary result to derive the eventual clusters. In addition to the linear-time iterative solvers, TPO further includes a greedy initialization trick for speeding up the convergence, and an attribute dimension reduction approach to conspicuously boost the practical efficiency of TPO over graphs with large attribute sets, without degrading result quality. Our empirical studies, which involved 5 real ABGs and compared against 19 existing algorithms, demonstrate that TPO consistently attains superior or comparable

Table 1: Frequently used notations.

Notation	Description
$\mathcal{U}, \mathcal{V}, \mathcal{E}$	The node sets $\mathcal{U}$, $\mathcal{V}$, and the edge set $\mathcal{E}$ of ABG $\mathcal{G}$.
$\mathbf{X}_{\mathcal{U}}, \mathbf{X}_{\mathcal{V}}$	Attribute vectors of nodes in $\mathcal{U}$ and $\mathcal{V}$.
$d_{\mathcal{U}}, d_{\mathcal{V}}$	Attribute dimensions of nodes in $\mathcal{U}$ and $\mathcal{V}$.
$w(u_i, v_j)$	Weight of edge (u_i, v_j) in $\mathcal{E}$.
k	The number of clusters.
α	Balance coefficient used in Eq. (3).
γ	Maximum number of iterations used in Eq. (31).
d	Dimension of $\mathbf{X}'_{\mathcal{U}}$ in Eq. (15) ($d \le d_{\mathcal{U}}$).
T_f, T_g	Maximum number of iterations used in Algorithms 2 and 3, respectively.
$\mathbf{L}_{\mathcal{U}}$	Normalized adjacency matrix defined in Eq. (7).
$\mathbf{Z}_{\mathcal{U}}, \widehat{\mathbf{Z}}_{\mathcal{U}}$	Feature vectors of nodes in $\mathcal{U}$ and their normalized version defined in Eq. (6) and Eq. (2), respectively.
$s(u_i, u_j)$	The MSA between nodes u_i and u_j defined in Eq. (1).
$\mathbf{R}$	Matrix satisfies $\mathbf{R}[i] \cdot \mathbf{R}[j] \approx s(u_i, u_j)$.
$\mathbf{Y}$	The NCI matrix defined in Eq. (10).
Υ	The continuous version of $\mathbf{Y}$ satisfying Eq. (11).

clustering quality at a fraction of the cost compared to the state-of-the-art methods. For instance, on the largest Amazon dataset with over 10 million nodes and 22 million edges, TPO obtains the best clustering accuracy within 3 minutes, whereas the state-of-the-art demands more than 4 hours to terminate.

2 Problem Formulation

2.1 Notation and Terminology

We denote matrices using bold uppercase letters, e.g., $\mathbf{M} \in \mathbb{R}^{n \times d}$, and the i-th row (resp. the j-th column) of $\mathbf{M}$ is represented as $\mathbf{M}[i]$ (resp. $\mathbf{M}[:, j]$). Accordingly, $\mathbf{M}[i, j]$ signifies the entry at the i-th row and j-th column of $\mathbf{M}$. For each vector $\mathbf{M}[i]$, we use $\|\mathbf{M}[i]\|$ to represent its L_2 norm and $\|\mathbf{M}\|_F$ to represent the Frobenius norm of $\mathbf{M}$.

Let $\mathcal{G} = (\mathcal{U} \cup \mathcal{V}, \mathcal{E}, \mathbf{X}_{\mathcal{U}}, \mathbf{X}_{\mathcal{V}})$ symbolize an *attributed bipartite graph* (ABG), where $\mathcal{E}$ is composed of edges connecting nodes in two disjoint node sets $\mathcal{U}$ and $\mathcal{V}$ and each edge (u_i, v_j) is associated with an edge weight $w(u_i, v_j)$. Each node $u_i \in \mathcal{U}$ (resp. $v_i \in \mathcal{V}$) of $\mathcal{G}$ is characterized by a length-$d_{\mathcal{U}}$ (resp. length-$d_{\mathcal{V}}$) attribute vector $\mathbf{X}_{\mathcal{U}}[i]$ (resp. $\mathbf{X}_{\mathcal{V}}[i]$). Further, we denote by $\mathbf{B}_{\mathcal{U}} \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{V}|}$ the adjacency matrix of $\mathcal{G}$ from the perspective of $\mathcal{U}$, in which $\mathbf{B}_{\mathcal{U}}[i, j] = w(u_i, v_j)$ if $(u_i, v_j) \in \mathcal{E}$ and 0 otherwise. Let $\mathbf{D}_{\mathcal{U}}$ (resp. $\mathbf{D}_{\mathcal{V}}$) be a $|\mathcal{U}| \times |\mathcal{U}|$ (resp. $|\mathcal{V}| \times |\mathcal{V}|$) diagonal matrix wherein the diagonal entry $\mathbf{D}_{\mathcal{U}}[i, i]$ (resp. $\mathbf{D}_{\mathcal{V}}[i, i]$) stands for the sum of the weights of edges incident to u_i (resp. v_i), i.e., $\sum_{(u_i, v_\ell) \in \mathcal{E}} w(u_i, v_\ell)$ (resp. $\sum_{(u_\ell, v_i) \in \mathcal{E}} w(u_\ell, v_i)$). Table 1 lists the frequently used notations throughout the paper.

The overarching goal of k-ABGC is formalized in Definition 2.1 and exemplified in Figure 1. Note that by default, we regard $\mathcal{U}$ as the target node set to cluster. The number k can be specified by users or configured by a preliminary procedure [41].

Definition 2.1 (k-Attributed Bipartite Graph Clustering (k-ABGC)). Given an ABG $\mathcal{G}$, the target node set $\mathcal{U}$, and the number k of clusters, k-ABGC aims to partition node set $\mathcal{U}$ into k disjoint clusters $\{C_1, C_2, \cdots, C_k\}$ such that nodes within the same cluster are close to each other in terms of both topological proximity and attribute
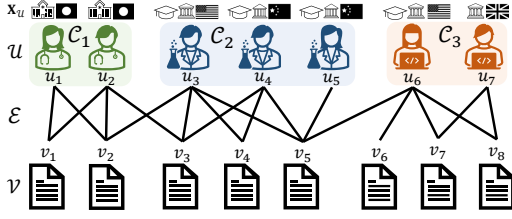
Figure 1: An Illustrative Example of k-ABGC

similarity while nodes across diverse clusters are distant.

2.2 Multi-Scale Attribute Affinity (MSA)

Notice that Definition 2.1 cannot directly guide the generation of clusters, as it lacks a concrete optimization objective that quantifies node affinities. To this end, we first delineate our novel affinity measure MSA for nodes in terms of both graph structure and attributes, before formally introducing our objective in Section 2.3.

MSA formulation. We first assume that each node $u_i \in \mathcal{U}$ can be represented by a feature vector $\mathbf{Z}_{\mathcal{U}}[i]$, which characterizes both the attributes as well as the rich semantics hidden in the bipartite graph topology. Following the popular Skip-gram model [40] and its extension to graphs [18, 47], we can model pair-wise affinity of nodes as a softmax unit [16] parametrized by a dot product of their feature vectors. Rather than using the vanilla softmax function, we adopt a symmetric softmax function and formulate the MSA $s(u_i, u_j)$ between any two nodes u_i, u_j in $\mathcal{U}$ as follows:

$$s(u_i, u_j) = \frac{e^{\widehat{Z}_{\mathcal{U}}[i] \cdot \widehat{Z}_{\mathcal{U}}[j]}}{\sqrt{\sum_{u_\ell \in \mathcal{U}} e^{\widehat{Z}_{\mathcal{U}}[i] \cdot \widehat{Z}_{\mathcal{U}}[\ell]}} \cdot \sqrt{\sum_{u_\ell \in \mathcal{U}} e^{\widehat{Z}_{\mathcal{U}}[j] \cdot \widehat{Z}_{\mathcal{U}}[\ell]}}}, \quad (1)$$

where $\widehat{Z}_{\mathcal{U}}$ is a normalized $\mathbf{Z}_{\mathcal{U}}$ whose each i-th row satisfies

$$\widehat{Z}_{\mathcal{U}}[i] = \mathbf{Z}_{\mathcal{U}}[i] / \|\mathbf{Z}_{\mathcal{U}}[i]\|. \quad (2)$$

MSA $s(u_i, u_j)$ is symmetric, i.e., $s(u_i, u_j) = s(u_j, u_i) \; \forall u_i, u_j \in \mathcal{U}$. Additionally, by imposing a normalization, $-1 \le \widehat{Z}_{\mathcal{U}}[i] \cdot \widehat{Z}_{\mathcal{U}}[j] \le 1$ $\forall u_i, u_j \in \mathcal{U}$, and hence, the MSA values w.r.t. any node $u_i \in \mathcal{U}$ are scaled to a similar range.

Optimization Objective for $\mathbf{Z}_{\mathcal{U}}$. Next, we focus on the obtainment of the feature vector $\widehat{Z}_{\mathcal{U}}[i]$ for each node $u_i \in \mathcal{U}$. A favorable choice might be graph neural networks (GNNs) [27], which, however, cannot be readily applied to ABGs as existing GNNs are primarily designed for general graphs, and it is rather costly to train classic GNNs. As demystified by recent studies [38, 70, 89], many popular GNNs models can be unified into an optimization framework from the perspective of numeric optimization, which essentially produces node feature vectors being smooth on nearby nodes in terms of the underlying graph. Inspired by this finding, we extend this optimization framework to ABGs. More specifically, its objective is as follows:

$$\min_{\mathbf{Z}_{\mathcal{U}}} (1 - \alpha) \cdot O_a + \alpha \cdot O_g, \quad (3)$$

which includes a non-negative coefficient $\alpha \in [0, 1]$ and two terms: (i) a fitting term O_a in Eq. (4) aiming at ensuring $\mathbf{Z}_{\mathcal{U}}$ is close to the input attribute vectors $\mathbf{X}_{\mathcal{U}}$,

$$O_a = \|\mathbf{Z}_{\mathcal{U}} - \mathbf{X}_{\mathcal{U}}\|_F^2 \quad (4)$$

and (ii) a regularization term O_g in Eq. (5) constraining the feature

vectors of two nodes with high connectivity to be similar.

$$O_g = \frac{1}{2} \sum_{u_i, u_j \in \mathcal{U}} \widehat{w}(u_i, u_j) \cdot \left\| \frac{\mathbf{Z}_{\mathcal{U}}[i]}{\sqrt{\mathbf{D}_{\mathcal{U}}[i, i]}} - \frac{\mathbf{Z}_{\mathcal{U}}[j]}{\sqrt{\mathbf{D}_{\mathcal{U}}[j, j]}} \right\|^2 \quad (5)$$

The regularization term requires scaling $\mathbf{Z}_{\mathcal{U}}[i]$ of each node u_i in Eq. (5) with a factor $1/\sqrt{\mathbf{D}_{\mathcal{U}}[i, i]}$ to avoid distorting the values in $\mathbf{Z}_{\mathcal{U}}[i]$ when u_i connects to massive or scant links. The weight $\widehat{w}(u_i, u_j)$ in Eq. (5) is defined by

$$\widehat{w}(u_i, u_j) = \sum_{v_\ell \in \mathcal{N}(u_i) \cap \mathcal{N}(u_j)} \frac{w(u_i, v_\ell) \cdot w(v_\ell, u_j)}{\mathbf{D}_{\mathcal{V}}[\ell, \ell]},$$

which reflects the strength of connections between two homogeneous nodes u_i and u_j in $\mathcal{U}$ (e.g., researchers) via their common neighbors in the counterparty $\mathcal{V}$ (e.g., co-authored papers). As an example for illustration, consider researchers u_3, u_4 in Figure 1, $\widehat{w}(u_3, u_4) = \frac{1}{3} + \frac{1}{2} + \frac{1}{4}$, where the denominators 3, 2, and 4 correspond to the numbers of authors in papers v_3, v_4 and v_5. Accordingly, $\widehat{w}(u_3, u_4)$ evaluates the overall contributions of u_3, u_4 to their collaborated research works in $\mathcal{V}$. Thus, the O_g term in Eq. (3) is to minimize the distance of feature vectors of researchers who have extensively collaborated with each other with high contributions.

The hyper-parameter α balances the attribute and topology information encoded into $\mathbf{Z}_{\mathcal{U}}$. In particular, when $\alpha = 0$, feature vectors $\mathbf{Z}_{\mathcal{U}} = \mathbf{X}_{\mathcal{U}}$, and at the other extreme, i.e., $\alpha = 1$, $\mathbf{Z}_{\mathcal{U}}$ is entirely dependent on the topology of $\mathcal{G}$.

Closed-form Solution of $\mathbf{Z}_{\mathcal{U}}$. Given an α, Lemma 2.2[1] indicates that the optimal feature vectors $\mathbf{Z}_{\mathcal{U}}$ to Eq. (3) can be computed via iterative sparse matrix multiplications in Eq. (6) without undergoing expensive training.

LEMMA 2.2. *When $\gamma \to \infty$, $\mathbf{Z}_{\mathcal{U}}$ in Eq. (6) is the closed-form solution to the optimization problem in Eq. (3).*

$$\mathbf{Z}_{\mathcal{U}} = (1 - \alpha) \sum_{r=0}^{\gamma} \alpha^r \cdot \left(\mathbf{L}_{\mathcal{U}} \mathbf{L}_{\mathcal{U}}^\top \right)^r \mathbf{X}_{\mathcal{U}}, \quad (6)$$

where $\mathbf{L}_{\mathcal{U}}$ is a normalized version of adjacency matrix $\mathbf{B}_{\mathcal{U}}$, i.e.,

$$\mathbf{L}_{\mathcal{U}} = \mathbf{D}_{\mathcal{U}}^{-\frac{1}{2}} \mathbf{B}_{\mathcal{U}} \mathbf{D}_{\mathcal{V}}^{-\frac{1}{2}}. \quad (7)$$

In practice, we set γ in Eq. (6) to a finite number (typically 5) for efficiency. Intuitively, the computation of $\mathbf{Z}_{\mathcal{U}}$ essentially aggregates attributes from other homogeneous nodes as per their multi-scale proximities (e.g., the strength of connections via multiple hops (at most γ hops)) in $\mathcal{G}$. As such, the feature vectors of nodes with numerous direct or indirect linkages will be more likely to be close, yielding a high MSA in Eq. (1).

2.3 Objective Function

Based on the foregoing definitions of k-ABGC and MSA, we formulate the problem of k-ABGC as the following objective function:

$$\min_{C_1, C_2, \cdots, C_k} \sum_{\ell=1}^{k} \frac{1}{|C_\ell|} \sum_{u_i \in C_\ell, u_j \in \mathcal{U} \setminus C_\ell} s(u_i, u_j), \quad (8)$$

More precisely, Eq. (8) is to identify k disjoint clusters $C_1, C_2, \cdots, C_k$ in $\mathcal{U}$ such that the average MSA of two nodes in different clusters is low. Meanwhile, with this optimization objective, the average

---

[1]All proofs appear in Appendix A.

MSA of any two nodes in the same cluster will be maximized; in other words, nodes within the same cluster are tight-knit.

According to [52], Eq. (8) is an NP-complete combinatorial optimization problem. Hence, the exact solution to Eq. (8) is computationally infeasible when $\mathcal{U}$ contains a large number of nodes. Moreover, the direct optimization of Eq. (8) demands materializing $s(u_i, u_j)$ of every node pairs in $\mathcal{U} \times \mathcal{U}$. As such, deriving an approximate solution by optimizing Eq. (8) with even a handful of epochs entails an $O(|\mathcal{E}| \cdot |\mathcal{U}| \cdot d_{\mathcal{U}})$ computational cost and a quadratic space overhead $O(|\mathcal{U}|^2)$, rendering it incompetent for large ABGs.

3 The TPO Algorithm

To address the above-said challenges, this section presents our *Three-Phase Optimization* framework (TPO) to k-ABGC computation based on Eq. (8) without explicitly constructing the MSA matrix.

3.1 Synoptic Overview

At a high level, TPO draws inspiration from the equivalence between the optimization objectives in Eq. (8) and Eq. (9), as Lemma 3.1.

LEMMA 3.1. *Eq. (8) is equivalent to the following objective:*
$$\min_{\mathbf{Y} \geq 0, \mathbf{H} \geq 0} \|\mathbf{R} - \mathbf{Y}\mathbf{H}^\top\|_F^2 \text{ s.t. } \mathbf{Y} \text{ is an NCI matrix.} \quad (9)$$

Specifically, if we can identify a matrix $\mathbf{R}$ such that $\mathbf{R}[i] \cdot \mathbf{R}[j] = s(u_i, u_j) \ \forall u_i.u_j \in \mathcal{U}$, the computation of k non-overlapping clusters $C_1, C_2, \cdots, C_k$ towards optimizing Eq. (8) is equivalent to decomposing $\mathbf{R}$ into two non-negative matrices $\mathbf{Y}$ and $\mathbf{H}$, where $\mathbf{Y}$ represents a *normalized cluster indicator* (NCI) matrix $\mathbf{Y} \in \mathbb{R}^{|\mathcal{U}| \times k}$, as defined in Eq. (10).
$$\mathbf{Y}[i, \ell] = \begin{cases} \frac{1}{\sqrt{|C_\ell|}} & \text{if } u_i \text{ belongs to in cluster } C_\ell, \\ 0 & \text{otherwise.} \end{cases} \quad (10)$$
According to Eq. (10), for each node $u_i \in \mathcal{U}$, its corresponding vector $\mathbf{Y}[i]$ in the NCI matrix comprises solely one non-zero entry $\mathbf{Y}[i, \ell]$ indicating the clustering membership of u_i, and the value should be $1/\sqrt{|C_\ell|}$. This characteristic ensures that $\mathbf{Y}$ is column-orthogonal, i.e., $\mathbf{Y}^\top \mathbf{Y} = \mathbf{I}$. However, this constraint on $\mathbf{Y}$ renders the factorization of $\mathbf{R}$ hard to converge. Instead of directly computing the exact $\mathbf{Y}$, we employ a two-step approximation strategy. More specifically, TPO first builds a $|\mathcal{U}| \times k$ matrix Υ (a continuous version of $\mathbf{Y}$) which minimizes the factorization loss in Eq. (11):
$$\min_{\Upsilon \geq 0, \mathbf{H} \geq 0} \|\mathbf{R} - \Upsilon \mathbf{H}^\top\|_F^2 \text{ s.t. } \Upsilon^\top \Upsilon = \mathbf{I}, \quad (11)$$
in which the constraint on $\mathbf{Y}$ in Eq. (9) is relaxed to be $\Upsilon \geq 0$ and $\Upsilon^\top \Upsilon = \mathbf{I}$. Afterward, the task is to transform Υ into an NCI matrix $\mathbf{Y}$ by minimizing their difference about Eq. (9).

As outlined in Figure 2, given an ABG $\mathcal{G}$, the number of k of clusters, and the node set $\mathcal{U}$ to be partitioned as input, TPO outputs an approximate solution to the k-ABGC problem in Eq. (8) through three phases: (i) constructing a low-dimensional matrix $\mathbf{R}$ such that $\mathbf{R}[i] \cdot \mathbf{R}[j] \approx s(u_i, u_j) \ \forall u_i.u_j \in \mathcal{U}$ without explicitly materializing the MSA of all node pairs (Algorithm 1, Section 3.2); (ii) factorizing $\mathbf{R}$ as per Eq. (11) to create a $\mathcal{U} \times k$ non-negative column-orthogonal matrix Υ (Algorithm 2, Section 3.3); and (iii) effectively converting Υ into an NCI $\mathbf{Y}$ (Algorithm 3, Section 3.4). In what follows, we elaborate on the algorithmic details of these three subroutines. Due to space limit, we defer the complexity analysis of them and TPO to
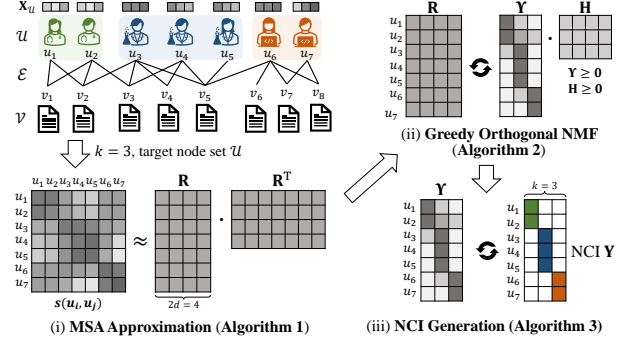

Figure 2: Overview of TPO

our technical report [78].

3.2 MSA Approximation via Random Features

Algorithm 1 illustrates the pseudo-code of linearizing the approximate computation of MSA in Eq. (1) as the matrix product $\mathbf{R} \cdot \mathbf{R}^\top$ with matrix $\mathbf{R}$. The fundamental idea is to leverage and tweak the *random features* [49, 82] technique designed for approximating the Gaussian kernel $e^{-\|\mathbf{x} - \mathbf{y}\|^2/2}$ of any vectors $\mathbf{x}$ and $\mathbf{y}$.

After taking as input the ABG $\mathcal{G}$ and parameters α, γ, Algorithm 1 begins by calculating $\mathbf{L}_{\mathcal{U}}$ according to Eq. (7) and initializing $\mathbf{Z}_{\mathcal{U}}$ as $\alpha \mathbf{X}_{\mathcal{U}}$ (Lines 1-2). At Lines 3-4, we update $\mathbf{Z}_{\mathcal{U}}$ via γ iterations of the following matrix multiplication:
$$\mathbf{Z}_{\mathcal{U}} \leftarrow \alpha \cdot (\mathbf{X}_{\mathcal{U}} + \mathbf{L}_{\mathcal{U}} \cdot (\mathbf{L}_{\mathcal{U}}^\top \mathbf{Z}_{\mathcal{U}})). \quad (12)$$
Particularly, we structure the matrix multiplication $\mathbf{L}_{\mathcal{U}} \mathbf{L}_{\mathcal{U}}^\top \mathbf{Z}_{\mathcal{U}}$ as $\mathbf{L}_{\mathcal{U}} \cdot (\mathbf{L}_{\mathcal{U}}^\top \mathbf{Z}_{\mathcal{U}})$ in Eq. (12) to boost the computation efficiency. Subsequently, Algorithm 1 transforms $\mathbf{Z}_{\mathcal{U}}$ into $\widehat{\mathbf{Z}}_{\mathcal{U}}$ by applying an L_2 normalization to each row in $\mathbf{Z}_{\mathcal{U}}$ (Line 5) and then proceeds to constructing $\mathbf{R}$ (Lines 6-9).

To be specific, we first generate a $d_{\mathcal{U}} \times d_{\mathcal{U}}$ Gaussian random matrix $\mathbf{G}$ with every entry sampled independently from the standard normal distribution (Line 6) and then apply a QR decomposition over it to get a $d_{\mathcal{U}} \times d_{\mathcal{U}}$ orthogonal matrix $\mathbf{Q}$ (Line 7). The matrix $\mathbf{Q}$ is distributed uniformly on the Stiefel manifold, i.e., the space of all orthogonal matrices [43]. Next, Algorithm 1 calculates $\mathbf{R}'$ by
$$\mathbf{R}' = \sqrt{\frac{e}{d_{\mathcal{U}}}} \cdot \left(sin(\widehat{\mathbf{Z}}_{\mathcal{U}}^\circ) \, \| \, cos(\widehat{\mathbf{Z}}_{\mathcal{U}}^\circ) \right) \in \mathbb{R}^{|\mathcal{U}| \times 2d_{\mathcal{U}}}, \quad (13)$$
where $\widehat{\mathbf{Z}}_{\mathcal{U}}^\circ = \sqrt{d_{\mathcal{U}}} \cdot \widehat{\mathbf{Z}}_{\mathcal{U}} \cdot \mathbf{Q}^\top$ and $\|$ represents the horizontal concatenation operator for matrices (Line 8). Finally, in Line 9, the matrix $\mathbf{R}$ is obtained by normalizing each row of $\mathbf{R}'$ as
$$\mathbf{R}[i] = \frac{\mathbf{R}'[i]}{\sqrt{\mathbf{R}'[i] \cdot \mathbf{r}}} \in \mathbb{R}^{2d_{\mathcal{U}}} \text{ where } \mathbf{r} = \sum_{u_\ell \in \mathcal{U}} \mathbf{R}'[\ell]. \quad (14)$$

THEOREM 3.2. *For any two nodes $u_i, u_j \in \mathcal{U}$, if $\mathbf{R}$ is the output of Algorithm 1, then the following inequality holds:*
$$\frac{1 - \frac{17}{16d_{\mathcal{U}}^2} - \frac{1}{4d_{\mathcal{U}}}}{1 + \frac{17}{16d_{\mathcal{U}}^2} + \frac{1}{4d_{\mathcal{U}}}} \cdot s(u_i, u_j) \leq \mathbb{E}[\mathbf{R}[i] \cdot \mathbf{R}[j]] \leq \frac{1 + \frac{17}{16d_{\mathcal{U}}^2} + \frac{1}{4d_{\mathcal{U}}}}{1 - \frac{17}{16d_{\mathcal{U}}^2} - \frac{1}{4d_{\mathcal{U}}}} \cdot s(u_i, u_j)$$

Theorem 3.2 indicates that $\mathbb{E}[\mathbf{R}[i] \cdot \mathbf{R}[j]]$ serves as an accurate estimator of $s(u_i, u_j)$, exhibiting extremely low bias, particularly because $d_{\mathcal{U}}$ often exceeds hundreds in practical scenarios.

Algorithm 1: MSA Approximation

Input: An ABG $\mathcal{G} = (\mathcal{U} \cup \mathcal{V}, \mathcal{E}, \mathbf{X}_{\mathcal{U}})$, target node set $\mathcal{U}$,
the balance coefficient α, the number γ of iterations
Output: Matrix $\mathbf{R}$

1 Calculate $\mathbf{L}_{\mathcal{U}}$ according to Eq. (7);
2 $\mathbf{Z}_{\mathcal{U}} \leftarrow \alpha \mathbf{X}_{\mathcal{U}}$;
3 **for** $r \leftarrow 1$ *to* γ **do**
4   Update $\mathbf{Z}_{\mathcal{U}}$ according to Eq. (12);
5 Normalize $\mathbf{Z}_{\mathcal{U}}$ as $\widehat{\mathbf{Z}}_{\mathcal{U}}$ by Eq. (2);
6 Sample a Gaussian random matrix $\mathbf{G} \in \mathbb{R}^{d_{\mathcal{U}} \times d_{\mathcal{U}}}$;
7 Compute $\mathbf{Q}$ by a QR decomposition over $\mathbf{G}$;
8 Compute $\mathbf{R}'$ according to Eq. (13);
9 **for** $u_i \in \mathcal{U}$ **do** Compute $\mathbf{R}[i]$ according to Eq. (14);
10 **return** $\mathbf{R}$;

3.2.1 **SVD-based Attribute Dimension Reduction.** Although Algorithm 1 circumvents the need to construct the MSA for all node pairs, it remains tenaciously challenging when dealing with ABGs with vast attribute sets, i.e., $d_{\mathcal{U}}$ being large. Recall that the major computation expenditure in Algorithm 1 lies at Lines 3-4 and Lines 7-8, which need $O(\gamma \cdot |\mathcal{E}| \cdot d_{\mathcal{U}})$ and $O(d_{\mathcal{U}}^3 + |\mathcal{U}| \cdot d_{\mathcal{U}}^2)$ time, respectively. As a result, when $d_{\mathcal{U}}$ is high, e.g., $d_{\mathcal{U}} = O(|\mathcal{U}|)$, the computational complexity of Algorithm 1 increases dramatically to be cubic, rendering it impractical for large-scale ABGs.

To address this, we refine the input attribute vectors $\mathbf{X}_{\mathcal{U}}$ by reducing their dimension from $d_{\mathcal{U}}$ to a much smaller constant d ($d \ll d_{\mathcal{U}}$). This approach aims to ensure that the d-dimensional approximation $\mathbf{X}'_{\mathcal{U}}$ of $\mathbf{X}_{\mathcal{U}}$ still accurately preserves the MSA as per Eq. (1). This adjustment reduces the computational cost to a linear time complexity of $O(\gamma \cdot |\mathcal{E}| + |\mathcal{U}|)$ since d is a constant. To realize this idea, we first apply the top-d *singular value decomposition* (SVD) over $\mathbf{X}_{\mathcal{U}}$ to produce the decomposition result $\mathbf{X}_{\mathcal{U}} \approx \Gamma \Sigma \Psi^{\top}$. Utilizing the column-orthogonal (semi-unitary) property of Ψ, i.e., $\Psi^{\top} \Psi = \mathbf{I}$, we have $\mathbf{X}_{\mathcal{U}} \mathbf{X}_{\mathcal{U}}^{\top} \approx \Gamma \Sigma \Psi^{\top} \Psi \Sigma \Gamma^{\top} = \Gamma \Sigma^2 \Gamma^{\top}$, implying

$$\mathbf{X}'_{\mathcal{U}} = \Gamma \Sigma \in \mathbb{R}^{|\mathcal{U}| \times d}, \tag{15}$$

which can be employed as a low-dimensional substitute of $\mathbf{X}_{\mathcal{U}}$ input to Algorithm 1. Along this line, we can derive a low-dimensional version $\mathbf{Z}'_{\mathcal{U}}$ of feature vectors $\mathbf{Z}_{\mathcal{U}}$ using the iterative process at Lines 2-4 in Algorithm 1 by simply replacing $\mathbf{X}_{\mathcal{U}}$ as $\mathbf{X}'_{\mathcal{U}}$, i.e.,

$$\mathbf{Z}'_{\mathcal{U}} = (1 - \alpha) \sum_{r=0}^{\infty} \alpha^r \cdot \left(\mathbf{L}_{\mathcal{U}} \mathbf{L}_{\mathcal{U}}^{\top} \right)^r \mathbf{X}'_{\mathcal{U}}.$$

LEMMA 3.3. *Let $\Gamma \Sigma \Psi^{\top}$ be the exact top-d SVD of $\mathbf{X}_{\mathcal{U}}$.*

$$\left| \mathbf{Z}'_{\mathcal{U}}[i] \cdot \mathbf{Z}'_{\mathcal{U}}[j] - \mathbf{Z}_{\mathcal{U}}[i] \cdot \mathbf{Z}_{\mathcal{U}}[j] \right| \leq \frac{\sigma_{d+1}^2 \sqrt{\mathbf{D}_{\mathcal{U}}[i,i] \cdot \mathbf{D}_{\mathcal{U}}[j,j]}}{1 - \alpha}$$

holds for every two nodes $u_i, u_j \in \mathcal{U}$, where σ_{d+1} is the $(d+1)$-th largest singular value of $\mathbf{X}_{\mathcal{U}}$.

Lemma 3.3 establishes the approximation guarantee of $\mathbf{Z}'_{\mathcal{U}}$, which theoretically assures the accurate approximation of the MSA defined in Eq. (1). Aside from the capabilities of preserving MSA and reducing computation load, this SVD-based trick can surprisingly denoise attribute data for enhanced clustering by its close connection to *principal component analysis* (PCA), as validated by our

---

Algorithm 2: Greedy Orthogonal NMF

Input: Matrix $\mathbf{R}$, the number k of clusters, the number T_f of iterations
Output: Matrix Υ

1 $\Gamma, \Sigma, \Psi \leftarrow$ RandomizedSVD$(\mathbf{R}, k)$;
2 Initialize Υ and $\mathbf{H}$ according to Eq. (18);
3 **for** $t \leftarrow 1$ *to* T_f **do**
4   Update $\mathbf{H}[j, \ell]$ $\forall 1 \leq j \leq 2d, 1 \leq \ell \leq k$ by Eq. (16);
5   Update $\Upsilon[i, \ell]$ $\forall u_i \in \mathcal{U}, 1 \leq \ell \leq k$ by Eq. (17);
6 **return** Υ;

---

experiments in Section 4.2.

3.3 Greedy Orthogonal NMF

Upon constructing $\mathbf{R} \in \mathbb{R}^{|\mathcal{U}| \times 2d}$ (with $d = d_{\mathcal{U}}$ if the dimension reduction from Section 3.2.1 is not applied) in Algorithm 1, TPO passes it to the second phase, i.e., conducting an orthogonal non-negative matrix factorization (NMF) of $\mathbf{R}$ as in Eq. (11) to create Υ. The pseudo-code of our solver to this problem is presented in Algorithm 2, iteratively updating Υ and $\mathbf{H}$ using an alternative framework towards optimizing the objective function in Eq. (11). (Lines 3-5). Specifically, given the number T_f of iterations and initial guess of $\mathbf{H}$ and Υ, in each iteration, we first update each (j, ℓ)-entry ($1 \leq j \leq 2d$ and $1 \leq \ell \leq k$) in $\mathbf{H}$ following Eq. (16) while fixing Υ, and then update $\Upsilon[i, \ell]$ for $u_i \in \mathcal{U}$ and $1 \leq \ell \leq k$ as in Eq. (17) with $\mathbf{H}$ fixed.

$$\mathbf{H}[j, \ell] = \mathbf{H}[j, \ell] \cdot \frac{(\mathbf{R}^{\top} \Upsilon)[j, \ell]}{(\mathbf{H} \cdot (\Upsilon^{\top} \Upsilon))[j, \ell]} \tag{16}$$

$$\Upsilon[i, \ell] = \Upsilon[i, \ell] \cdot \sqrt{\frac{(\mathbf{RH})[i, \ell]}{(\Upsilon \cdot (\Upsilon^{\top} \cdot (\mathbf{RH})))[i, \ell]}} \tag{17}$$

The above update rules for solving Eq. (11) can be derived by utilizing the *auxiliary function approach* [32] with Lagrangian multipliers in convex optimization, whose convergence is guaranteed by the monotonicity theorem [10]. Note that we reorder the matrix multiplications $\mathbf{H} \Upsilon^{\top} \Upsilon$ and $\Upsilon \Upsilon^{\top} \mathbf{RH}$ in Eq. (16) and (17) to $\mathbf{H} \cdot (\Upsilon^{\top} \Upsilon)$ and $\Upsilon \cdot (\Upsilon^{\top} \cdot (\mathbf{RH}))$, respectively, so as to avert materializing $2d \times |\mathcal{U}|$ dense matrix $\mathbf{H} \Upsilon^{\top}$ and $|\mathcal{U}| \times |\mathcal{U}|$ dense matrix $\Upsilon \Upsilon^{\top}$. As such, the computational complexities of updating $\mathbf{H}$ and Υ in Eq. (16) and (17) are reduced to $O(|\mathcal{U}|dk + |\mathcal{U}|k^2)$ per iteration.

The aforementioned computation is still rather costly due to the numerous iterations needed for the convergence of Υ and $\mathbf{H}$, especially when Υ and $\mathbf{H}$ are initialized randomly. We resort to a greedy seeding strategy to expedite convergence, as in many optimization problems. That is, we carefully select a good initialization of Υ and $\mathbf{H}$ in a fast but theoretically grounded manner. As described in Lines 1-2 in Algorithm 1, we set Υ and $\mathbf{H}$ as follows:

$$\Upsilon = \Gamma, \quad \mathbf{H} = \Psi \Sigma, \tag{18}$$

where Γ and Ψ are the top-k left and right singular vectors of $\mathbf{R}$, respectively, and Σ is a diagonal matrix whose diagonal entries are top-k singular values of $\mathbf{R}$, which are obtained by invoking the *truncated* randomized SVD algorithm [19] with $\mathbf{R}$ and k. Note that this routine consumes $O(|\mathcal{U}|dk + (\mathcal{U} + d)k^2)$ time [19] and can be done efficiently in practice in virtue of its randomized algorithmic design as well as the highly-optimized libraries (LAPACK and BLAS) for matrix operations under the hood.

Algorithm 3: Effective NCI Generation

Input: Matrix Υ and the number T_g of iterations
Output: The NCI matrix Y

1   $\Phi = I$;
2   **for** $t \leftarrow 1$ *to* T_g **do**
3   **for** $u_i \in \mathcal{U}$ **do**
4   Determine ℓ^* by Eq. (21);
5   Update Y by Eq. (22);
6   Normalize Y such that each column has a unit L_2 norm;
7      $\Phi \leftarrow \Upsilon^\top Y$;
8   **return** Y;

Given the fact that singular vectors $\Upsilon = \Gamma$ are column-orthogonal, i.e., $\Upsilon^\top \Upsilon = I$, the Eckart-Young Theorem [15] (Theorem A.1 in Appendix A) pinpoints that Eq. (18) offers the optimal solution to Eq. (11) when the non-negative constraints over Υ and H are relaxed. In simpler terms, Eq. (18) immediately gains a rough solution to our optimization objective in Eq. (11), thereby drastically curtailing the number of iterations needed for Lines 3-5.

3.4 Effective NCI Generation

In its final stage, TPO generates an NCI matrix Y by minimizing the "difference" between Υ returned by Algorithm 2 and the target NCI matrix Y. Recall from Eq. (9), our original objective is to find a $|\mathcal{U}| \times k$ NCI matrix Y and a $2d \times k$ non-negative H such that the total squared reconstruction error $\|R - YH^\top\|_F^2 = \sum_{u_i \in \mathcal{U}} \sum_{j=1}^{d} (R[i, j] - Y[i] \cdot H[j])^2$ is minimized. Considering Υ is a continuous version of Y (relaxing the constraint in Eq. (10)), $\|R - \Upsilon H^\top\|_F^2$ is capable of attaining a strictly lower reconstruction error compared to $\|R - YH^\top\|_F^2$. Therefore, an ideal solution Y to Eq. (9) ensures that $\|R - YH^\top\|_F^2$ closely approximates $\|R - \Upsilon H^\top\|_F^2$ in Eq. (11). Mathematically, the conversion from matrix Υ into the NCI matrix Y can be formulated as the minimization of the difference of their reconstruction errors, i.e., $\left| \|R - YH^\top\|_F^2 - \|R - \Upsilon H^\top\|_F^2 \right| = \left| \text{trace}((YY^\top - \Upsilon\Upsilon^\top) \cdot RR^\top) \right|$ by Lemma 3.4.

LEMMA 3.4. *The following equation holds:*

$$\left| \|R - YH^\top\|_F^2 - \|R - \Upsilon H^\top\|_F^2 \right| = \left| \text{trace}\big((YY^\top - \Upsilon\Upsilon^\top) \cdot RR^\top\big) \right|. \quad (19)$$

Further, we reformulate the problem as follows:

$$\min_{\Phi, Y} \|\Upsilon\Phi - Y\|_2 \text{ s.t. } \Phi\Phi^\top = I \text{ and } Y \text{ is an NCI matrix,} \quad (20)$$

which implies that, if the NCI matrix Y and the $k \times k$ row-orthogonal matrix Φ minimize $\|\Upsilon\Phi - Y\|_2$, $YY^\top - \Upsilon\Upsilon^\top \approx \Upsilon\Phi\Phi^\top\Upsilon^\top - \Upsilon\Upsilon^\top \approx 0$ holds and the objective loss in Eq. (19) is therefore minimized.

To solve Eq. (20), we develop Algorithm 3 in TPO, which obtains the NCI matrix Y through an iterative framework wherein Φ and Y are refined in an alternative fashion till convergence. Initially, Algorithm 3 starts by taking as input the matrix Υ and the number T_g of iterations and initializing Φ as a $k \times k$ identity matrix (Line 1). It then launches an iterative process at Lines 2-7 to jointly refine Y and Φ. Specifically, in each of the T_g iterations, TPO first determines the cluster id of each node $u_i \in \mathcal{U}$ via (Line 4)

$$\ell^* = \arg \max_{1 \leq \ell \leq k} \Upsilon[i] \cdot \Phi[:, \ell] \quad (21)$$

and then updates the cluster indicator $Y[i]$ of node u_i as follows

Table 2: Attributed Bipartite Graphs

Name	CiteSeer	Cora	MovieLens	Google	Amazon
$|\mathcal{U}|$	1,237	1,312	6,040	64,527	2,330,066
$|\mathcal{V}|$	742	789	3,883	868,937	8,026,324
$|\mathcal{E}|$	1,665	2,314	1,000,209	1,487,747	22,507,155
$d_{\mathcal{U}}$	3,703	1,433	30	1,024	800
$d_{\mathcal{V}}$	3,703	1,433	21	-	-
k	6	7	21	5	3

(Line 5):

$$Y[i, \ell] = \begin{cases} 1 & \text{if } \ell = \ell^*, \\ 0 & \text{otherwise,} \end{cases} \quad \forall 1 \leq \ell \leq k. \quad (22)$$

Each column in Y is later L_2-normalized, i.e.,

$$\forall 1 \leq \ell \leq k \quad \sqrt{\sum_{u_i \in \mathcal{U}} Y[i, \ell]^2} = 1, \quad (23)$$

in accordance with the NCI constraint in Eq. (10) (Line 6). In a nutshell, Lines 3-6 optimizes Eq. (20) by updating Y with Φ fixed. To explain, recall the constraint of the NCI matrix Y stated in Eq. (10), each row of Y has merely one non-zero entry. Hence, by locating the column id ℓ^* whose corresponding entry $(\Upsilon\Phi)[i, \ell^*]$ is maximum in the i-th row of $\Upsilon\Phi$ (i.e., Eq. (21)) and meanwhile updating $Y[i]$ as Eqs. (22) and (23) as Lines 5-6, the distance between $\Upsilon\Phi$ and Y in Eq. (20) is naturally minimized.

With the refined Y at hand, the subsequent work turns into updating the $k \times k$ matrix Φ towards optimizing

$$\min_{\Phi} \|\Upsilon\Phi - Y\|_2 \text{ s.t. } \Phi\Phi^\top = I.$$

Given Y, the minimizer to this problem is $\Phi = \Upsilon^\top Y$ by utilizing Lemma 4.14 in [61]. Therefore, Φ is updated to $\Upsilon^\top Y$ at Line 7.

After repeating the above procedure for T_g iterations, TPO returns Y as the final clustering result. Practically, a dozen iterations are sufficient to yield high-caliber Y, as validated in Section 4.3.

4 Experiments

In this section, we experimentally evaluate our proposed k-ABGC method TPO against 19 competitors over five real ABGs in terms of clustering quality and efficiency. All the experiments are conducted on a Linux machine powered by 2 Xeon Gold 6330 @2.0GHz CPUs and 1TB RAM. For reproducibility, the source code and datasets are available at https://github.com/HKBU-LAGAS/TPC.

4.1 Experimental Setup

Datasets. Table 2 lists the statistics of the five datasets used in the experimental study. $|\mathcal{U}|$, $|\mathcal{V}|$, and $|\mathcal{E}|$ denote the cardinality of two disjoint node sets $\mathcal{U}, \mathcal{V}$, and edge set $\mathcal{E}$ of $\mathcal{G}$, respectively, while $d_{\mathcal{U}}$ (resp. $d_{\mathcal{V}}$) stands for the dimensions of attribute vectors of nodes in $\mathcal{U}$ (resp. $\mathcal{V}$). The number of ground-truth clusters of nodes $\mathcal{U}$ in $\mathcal{G}$ is k. Citeseer and Cora are synthesized from real citation graphs in [27] by dividing nodes in each cluster into two equal-sized partitions (i.e., $\mathcal{U}$ and $\mathcal{V}$) and removing intra-partition edges and isolated nodes as in [62]. In particular, nodes represent publications, edges denote their citation relationships, and labels correspond to the fields of study. The well-known MovieLens dataset [20] comprises user-movie ratings, where clustering labels are users' occupations in $\mathcal{U}$. Google and Amazon are extracted from the Google Maps [68]

Table 3: Clustering Quality (Larger ACC, NMI, and ARI indicate higher clustering quality).

Method	CiteSeer			Cora			MovieLens			Google			Amazon			Rank
	ACC	NMI	ARI	ACC	NMI	ARI	ACC	NMI	ARI	ACC	NMI	ARI	ACC	NMI	ARI	
KMeans [21]	0.526	0.277	0.225	0.431	0.229	0.137	0.298	0.363	0.170	0.370	0.053	0.012	0.502	0.038	0.079	5.4
SpecClust [56]	0.222	0.017	-0.001	0.311	0.026	0.003	0.318	0.392	0.197	-	-	-	-	-	-	11.27
NMF [65]	0.508	0.222	0.228	0.380	0.165	0.110	0.568	0.611	0.482	0.384	0.069	0.062	0.390	0.006	0.015	5.6
SCC [8]	0.243	0.015	0.012	0.280	0.040	0.017	0.128	0.030	0.004	0.425	0.038	0.038	0.470	0.018	-0.016	10.27
SBC [28]	0.239	0.015	0.012	0.262	0.059	0.023	0.116	0.035	0.006	0.394	0.006	-0.003	0.485	0.003	0.005	10.87
CCMOD [2]	0.200	0.010	0.003	0.264	0.066	0.040	0.141	0.029	0.010	OOM	OOM	OOM	OOM	OOM	OOM	12.27
SpecMOD [29]	0.238	0.012	0.003	0.290	0.023	-0.007	0.125	0.033	0.009	OOM	OOM	OOM	OOM	OOM	OOM	12.6
InfoCC [9]	0.235	0.013	0.007	0.223	0.035	0.018	0.095	0.036	0.007	0.277	0.008	0.008	0.378	0.007	0.003	11.54
DeepCC [63]	0.205	0.013	0.004	0.213	0.014	0.006	0.093	0.027	0.004	0.324	0.105	0.031	-	-	-	12.67
HOPE [72]	0.473	0.169	0.288	0.268	0.025	0.043	0.115	0.009	0.037	0.269	0	0	0.452	0	0.002	
ACMin [77]	0.450	0.143	0.141	0.650	0.470	0.410	0.122	0.032	0.009	0.312	0.023	0.020	0.428	0.012	0.003	7.67
GRACE [12]	0.469	0.209	0.199	0.351	0.136	0.103	0.298	0.249	0.195	0.420	0	0	0.427	0.008	0	7.47
AGCC [85]	0.448	0.144	0.153	0.650	0.517	0.406	0.538	0.589	0.480	OOM	OOM	OOM	OOM	OOM	OOM	7.34
Dink-Net [37]	0.308	0	0	0.231	0.004	0.007	0.123	0.001	0	0.429	0	0	OOM	OOM	OOM	
node2vec [18]	0.209	0.007	0.001	0.220	0.008	0	0.074	0.011	0	0.280	0	0	-	-	-	14.8
BiNE [13]	0.196	0.005	0	0.174	0.005	-0.002	0.071	0.012	0	-	-	-	-	-	-	15.54
GEBE [73]	0.231	0.013	0.002	0.293	0.014	0.006	0.095	0.014	-0.002	0.428	0	0	0.489	0	0	12.14
PANE [75, 76]	0.443	0.154	0.136	0.537	0.526	0.339	0.855	0.923	0.838	0.359	0.127	0.070	0.497	0.083	0.102	4.2
BiANE [23]	0.259	0.057	0.016	0.341	0.239	0.080	0.091	0.053	0.013	-	-	-	-	-	-	10.21
TPO ($d = d_{\mathcal{U}}$)	0.541	0.256	0.265	0.662	0.477	0.408	0.931	0.961	0.957	0.367	0.112	0.091	0.502	0.045	0.091	2.54
TPO	0.625	0.322	0.348	0.671	0.475	0.416	0.931	0.961	0.957	0.444	0.135	0.138	0.504	0.055	0.104	1.27

and Amazon review dataset [22], where edges represent the reviews on restaurants and books posted by users.

Competitors and Parameters. We compare TPO against 19 existing methods, which can be categorized into four groups as follows:

- *Data Clustering*: KMeans [21], NMF [65], and SpecClust [56];
- *Network Embedding*: node2vec [18], BiNE [13], BiANE [23], PANE [75, 76], and GEBE [73];
- *Attributed Graph Clustering*: AGCC [85], ACMin [77], GRACE [12], Dink-Net [37];
- *Bipartite Graph Clustering*: SCC [8], SBC [28], InfoCC [9], SpecMOD [29], CCMOD [2], DeepCC [63], and HOPE [72].

Unless otherwise specified, on all datasets, we set the numbers T_f and T_g of iterations required by Algorithms 2 and 3 in our proposed TPO to 5 and 20, respectively. Regarding parameters α and γ, by default, we set $\alpha = 0.6$, $\gamma = 6$ on *CiteSeer* and *MovieLens*, $\alpha = 0.9$, $\gamma = 10$ on *Cora* and *Google*, and $\alpha = 0.5$, $\gamma = 1$ on *Amazon*, respectively. To deal with the high attribute dimensions $d_{\mathcal{U}}$ of the *CiteSeer*, *Cora*, *Google*, and *Amazon* datasets, we set their new attribute dimensions d in Section 3.2.1 to 32, 128, 32, and 64, respectively. We refer to the version of TPO *without* the attribute dimension reduction module in Section 3.2.1 as TPO ($d = d_{\mathcal{U}}$). More implementation details of our method and baselines are in our technical report [78].

Evaluation Metrics. Following convention, we adopt three widely used measures [6, 25, 31, 57, 77, 85] to assess the clustering quality, namely (i) *Clustering Accuracy* (ACC), (ii) *Normalized Mutual Information* (NMI), and (3) *Adjusted Rand Index* (ARI), for measuring the quality of clusters produced by each evaluated method in the presence of the ground-truth clusters of the tested dataset. Particularly, ACC and NMI scores range from 0 to 1.0, whilst ARI ranges from −0.5 to 1.0. For each of these metrics, higher values indicate better clustering quality. Regarding efficiency evaluation, we report the running time in seconds (measured in wall-clock time) of each method on each dataset, excluding the time for input (loading datasets) and output (saving clustering results). The formulas for evaluation metrics are in our technical report [78].

4.2 Clustering Performance

This set of experiments reports the clustering quality achieved by TPO and all competitors over the five datasets, as well as their respective running times. We omit a method if it cannot report the results within three days or incur out-of-memory (OOM) errors. Since TPO is randomized, we repeat it five times and report the average performance.

Clustering Quality. Table 3 shows the ACC, NMI, and ARI scores of all methods on five ABGs, and their overall average performance rankings. We highlight the top-3 best clustering results on each dataset in gray with darker shades indicating higher quality. TPO consistently outperforms the 17 competitors on the *CiteSeer*, *MovieLens*, and *Google* datasets in terms of ACC, NMI, and ARI, by substantial margins of up to 9.9% for ACC, 4.5% for NMI, and 12% for ARI, respectively. The only exceptions are on *Cora* and *Amazon*, where TPO achieves the highest ACC and ARI results but inferior NMI scores compared to PANE or AGCC. In addition, TPO ($d = d_{\mathcal{U}}$) exhibits competitive clustering effectiveness, which either is second only to TPO or obtains the third best clustering results in most cases. Specifically, TPO ($d = d_{\mathcal{U}}$) is comparable to TPO on *Cora*, *MovieLens*, and *Amazon* with a performance degradation at most 0.9% in ACC, 1.0% in NMI, and 1.3% in ARI. Over all datasets, TPO and TPO ($d = d_{\mathcal{U}}$) attain the best and second best average performance rank (smaller rank is better), respectively. The evident superiority of TPO and TPO ($d = d_{\mathcal{U}}$) manifests the accuracy of our proposed MSA model in Section 2.2 in preserving the attribute similarity and topological connections between nodes, as well as the effectiveness of theoretically-grounded three-phase optimization framework introduced in Section 3.

At this point, a keen reader may wonder why TPO with attribute dimension reduction outperforms TPO ($d = d_{\mathcal{U}}$) on most datasets, especially *CiteSeer* and *Google*, as it seems that the former is an approximate version of the latter. Notice that TPO and TPO ($d = d_{\mathcal{U}}$) output identical results, as dimension reduction is not needed on *MovieLens* and TPO turns to be TPO ($d = d_{\mathcal{U}}$). Recall that the only difference between TPO and TPO ($d = d_{\mathcal{U}}$) is that TPO employs a truncated SVD over the input attribute vectors $\mathbf{X}_{\mathcal{U}}$ of a node in $\mathcal{U}$
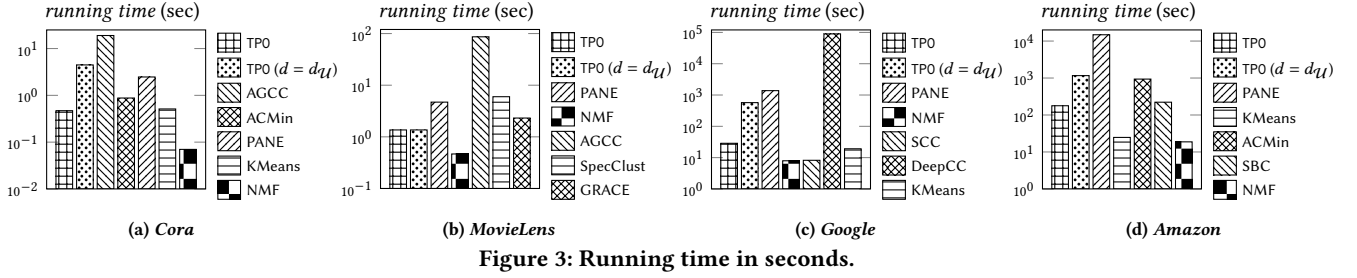
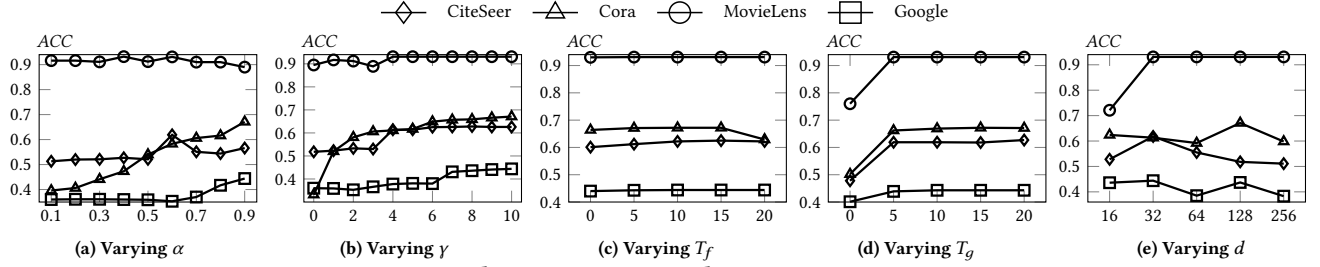**Figure 3: Running time in seconds.**

Figure 4: Clustering accuracy when varying parameters.

for dimension reduction as stated in Section 3.2.1. Aside from the crucial theoretical assurance offered by this SVD-based approach in the MSA approximation, it implicitly conducts a PCA on the attribute vectors, extracting key features from the input attributes while eradicating noisy ones. In brief, the SVD-based trick in Section 3.2.1 grants TPO the additional ability to denoise the attribute data, thus elevating the results' quality.

Efficiency. For clarity, we compare the empirical efficiency of TPO and TPO ($d = d_{\mathcal{U}}$) only against competitors ranked in the top 7 for clustering quality, as shown in Table 3. Figure 3 plots the computation times required by each of these methods on *Cora*, *MovieLens*, *Google*, and *Amazon*. The y-axis is the running time (seconds) in the log scale. On each of the diagrams in Figures 3(a), 3(b), 3(c), and 3(d), all the bars are displayed from left to right in an ascending order w.r.t. their average performance rank in Table 3. Accordingly, except the first two bars from the left for TPO and TPO ($d = d_{\mathcal{U}}$), the third bars (from the left) in these figures illustrate the running times of the best competitors, i.e., AGCC on *Cora*, and PANE on *MovieLens*, *Google*, and *Amazon*, respectively. As we can see, TPO is consistently faster than the state-of-the-art approaches, AGCC or PANE, on four datasets, often by orders of magnitude. For instance, on *Cora*, *Google*, and *Amazon*, TPO takes 0.47, 28.7, and 178 seconds, respectively, whereas the best baselines AGCC or PANE cost around 19 seconds, 23 minutes, and 4.1 hours, respectively, attaining 40×, 48×, and 83× speedup. In addition, TPO also enjoys a considerable efficiency gain of up to 19.9× over TPO ($d = d_{\mathcal{U}}$), attributed to the SVD-based dimension reduction (Section 3.2.1). On the *MovieLens* dataset, the input attribute dimension $d_{\mathcal{U}} = 30$ is low, and the attribute dimension reduction is therefore disabled, making TPO and TPO ($d = d_{\mathcal{U}}$) yield the same running time, which is 3.46× over the best competitor PANE. Although NMF, KMeans, and SCC run much faster than TPO on some datasets by either neglecting the graph topology or discarding the attribute data, their result quality is no match for our solution TPO.

In summary, TPO consistently delivers superior results for k-ABGC tasks over ABGs with various volumes while offering high practical efficiency, which corroborates the efficacy of our novel objective function based on MSA in Section 2 and the optimization solver with careful algorithmic designs developed in Section 3.

4.3 Parameter Analysis

In these experiments, we empirically investigate the impact of five key parameters in TPO: α, γ, T_f, T_g, and d. For each of them, we run TPO over *CiteSeer*, *Cora*, *MovieLens*, and *Google*, respectively, by varying the parameter with others fixed as in Section 4.1.

Varying α and γ. Figures 4(a) shows that on *Cora* and *Google*, TPO's clustering performance markedly improves as α increases from 0.1 to 0.9, indicating the importance of graph structure in these datasets. On *CiteSeer* and *MovieLens*, setting $\alpha = 0.6$ will be a favorable choice, which results in an optimal combination of attributes and graph topology and hence the highest ACC scores. Figures 4(b) depicts the ACC scores when γ increases from 0 to 10. When $\gamma = 0$, the graph structure is disregarded in TPO, namely $\mathbf{Z}_{\mathcal{U}} = \mathbf{X}_{\mathcal{U}}$. It can be observed on all datasets that the clustering quality rises with γ increasing except *CiteSeer* and *MovieLens*, where the ACC results reach a plateau after $\gamma \geq 6$. This is consistent with the fact that a larger γ produces a more accurate solution $\mathbf{Z}_{\mathcal{U}}$ to the objective in Eq. (3), and thus, higher clustering quality.

Varying T_f and T_g. Figures 4(c) presents the ACC scores when the T_f of iterations in Algorithm 2 is varied from 0 to 20. We can conclude that our greedy seeding strategy described in Section 3.3 is highly effective in enabling swift convergence, as additional optimization iterations merely bring minor gains in clustering performance. On *Cora* and *CiteSeer*, the ACC scores see an uptick when varying γ from 0 to 10, followed by a pronounced downturn. Such a performance decline is caused by overfitting in solving Eq. (11). From Figures 4(d) reporting clustering performance changes when

varying T_g from 0 to 20, we can make analogous observations on the four datasets. The evaluation scores first experience a sharp increase as T_g increases from 0 to 5. After that, the ACC remain invariant with γ increasing. The results manifest the effectiveness of our solver developed in Section 3.4 in fast NCI generation.

Varying d. Intuitively, a large d may lead to accurate preservation of MSA as per Lemma 3.3 and further improve clustering quality. However, in practice, the original attribute vectors $\mathbf{X}_{\mathcal{U}}$ embody noises, especially when $d_{\mathcal{U}}$ is high. As pinpointed and validated in Sections 3.2.1 and 4.2, our SVD-based dimension reduction inherently applies a PCA over $\mathbf{X}_{\mathcal{U}}$ for noise elimination, considerably upgrading the empirical result quality. That is to say, the choice of d strikes a balance between capturing MSA and removing noisy data, consistent with our empirical results in Figure 4(e). In particular, on *CiteSeer*, *Cora*, and *Google*, picking 32, 128, and 32 for dimension d, respectively, can strike a good balance between MSA preservation and noisy reduction for superior clustering performance. On *MovieLens*, the attribute dimension reduction is not enabled when $d \geq 32$ since its original dimension $d_{\mathcal{U}} = 30$. We refer interested readers to our technical report [78] for NMI and efficiency results.

5 Related Work

Bipartite Graph Clustering. A classic methodology [87] for bipartite graph clustering first projects a bipartite graph $\mathcal{G}$ into a unipartite graph by connecting every two nodes from the same partition $\mathcal{U}$ if they share common neighbors in $\mathcal{G}$. Then, a standard graph clustering algorithm for node clustering can be adopted on the constructed unipartite graph. However, the projection often leads to unipartite graphs $O(|\mathcal{U}|^2)$ edges, which is intolerable for even medium-sized graphs [71]. In our previous work [72], we addressed this problem by transforming it into a two-stage approximation framework.

Unlike the projection-based methods, another line of research focuses on simultaneously clustering two disjoint sets of nodes (i.e., $\mathcal{U}$ and $\mathcal{V}$) in a bipartite graph. These co-clustering techniques have been extensively investigated in the literature [17] and span a variety of applications in bioinformatics and text mining. Several attempts [8, 28, 29] are made to extend spectral clustering to bipartite graphs. Analogously, Ailem et al. [2] and Dhillon et al. [9] propose generating co-clusters by extending and optimizing classic metrics of modularity and mutual information on bipartite graphs, respectively. DeepCC [63] creates low-dimension instances and features using a deep autoencoder, then assigns clusters using a variant of the Gaussian mixture model. To handle the resolution limit in prior works as well as incorporate attribute information, Kim et al. [25] designed ABC, which incurs a severe efficiency issue due to its quadratic running time $O(|\mathcal{U}|^2 + |\mathcal{V}|^2)$.

Attributed Graph Clustering. As surveyed in [4, 5, 33, 77], there is a large body of work on attributed graph clustering (AGC). According to [77], existing AGC techniques can be categorized into four groups: edge-weigh-based methods [51, 79], distance-based methods [11, 88], statistics-based models [66, 77, 80], and graph learning-based methods [12, 42, 57, 58, 85]. Among them, graph learning-based approaches [12, 37, 42, 85] have achieved state-of-the-art performance, as reported in [30, 77]. These methods obtain high clustering quality on attributed graphs at the cost of costly neural network training, thus incurring poor scalability on large graphs. To our knowledge, the statistical-model-based solution, ACMin [77], is the only AGC method that scales to massive graphs with millions of nodes and billions of edges, while attaining high result quality. However, none of them are custom-made for ABGs, producing compromised result quality for k-ABGC.

Network Embedding. In recent years, network embedding, which converts each node in a graph into an embedding vector capturing the surrounding structures, has been employed in a wide range of graph analytics tasks, and has seen remarkable success [7, 14]. In particular, by simply feeding them into data clustering methods, e.g., KMeans, such embedding vectors can be utilized to cope with k-ABGC. However, the majority of network embedding works [13, 18, 47, 48, 53, 54, 73, 74] are designed for graphs in the absence of node attributes. To bridge this gap, a series of efforts [6, 24, 35, 44, 55, 60, 75, 76] have been made towards incorporating node attributes into embedding vectors for enhanced result utility. These approaches still suffer from sub-optimal clustering performance as they fall short of preserving the hidden semantics underlying bipartite graphs. To learn effective node embeddings over ABGs, [1, 23] extend SkipGram models [39] to ABGs by picking node-pair samples with consideration of both their intra-partition/inter-partition proximities and attribute similarities. Athar et al. [3] project the ABG into two homogeneous graphs based on topological connections and attribute similarities, and then invoke unsupervised GNNs on the constructed graphs for embedding generation. Moreover, Zhang et al. [86] propose IGE [86] for learning node embeddings on dynamic ABGs with a focus on temporal dependence of edges rather than the bipartite graph structures. These works either fall short of preserving multi-hop relationships between nodes or struggle to cope with large ABGs due to the significant expense of training.

6 Conclusion

This paper presents TPO, an effective and efficient solution for k-ABGC tasks. TPO achieves remarkable performance, attributed to a novel problem formulation based on the proposed multi-scale attribute affinity measure for nodes in ABGs, and a well-thought-out three-phase optimization framework for solving the problem. Through a series of theoretically-grounded efficiency techniques developed in this paper, TPO is able to scale to large ABGs with millions of nodes and hundreds of millions of edges while offering state-of-the-art result quality. The superiority of TPO over 19 baselines is experimentally validated over 5 real ABGs in terms of both clustering quality and empirical efficiency.

A   Theorems and Proofs

THEOREM A.1 (**Eckart–Young Theorem [15]**). *Suppose that $\mathbf{M}_k \in \mathbb{R}^{n \times k}$ is the rank-k approximation to $\mathbf{M} \in \mathbb{R}^{n \times n}$ obtained by exact SVD, then $\min_{rank(\widehat{\mathbf{M}}) \leq k} \|\mathbf{M} - \widehat{\mathbf{M}}\|_2 = \|\mathbf{M} - \mathbf{M}_k\|_2 = \sigma_{k+1}$, where σ_i represents the i-th largest singular value of $\mathbf{M}$.*

Proof of Lemma 2.2. We first rewrite that Eq. (5) as

$$\text{trace}(\mathbf{Z}_{\mathcal{U}}^{\top} \cdot (\mathbf{I} - \mathbf{L}_{\mathcal{U}} \mathbf{L}_{\mathcal{U}}^{\top}) \cdot \mathbf{Z}_{\mathcal{U}}). \tag{24}$$

The equivalence can be deduced by the definition of Eq. (5),

$$O_g = \frac{1}{2} \sum_{u_i, u_j \in \mathcal{U}} \widehat{w}(u_i, u_j) \cdot \left\| \frac{\mathbf{Z}_{\mathcal{U}}[i]}{\sqrt{\mathbf{D}_{\mathcal{U}}[i,i]}} - \frac{\mathbf{Z}_{\mathcal{U}}[j]}{\sqrt{\mathbf{D}_{\mathcal{U}}[j,j]}} \right\|^2$$

$$= \sum_{u_i, u_j \in \mathcal{U}} \frac{\widehat{w}(u_i, u_j)}{2} \left(\frac{\|\mathbf{Z}_{\mathcal{U}}[i]\|^2}{\mathbf{D}_{\mathcal{U}}[i,i]} + \frac{\|\mathbf{Z}_{\mathcal{U}}[j]\|^2}{\mathbf{D}_{\mathcal{U}}[j,j]} - \frac{2\mathbf{Z}_{\mathcal{U}}[i] \cdot \mathbf{Z}_{\mathcal{U}}[j]}{\sqrt{\mathbf{D}_{\mathcal{U}}[i,i]}\sqrt{\mathbf{D}_{\mathcal{U}}[j,j]}} \right)$$

$$= \sum_{v_\ell \in \mathcal{V}} \sum_{u_i, u_j \in \mathcal{U}} \frac{w(u_i, v_\ell) w(u_j, v_\ell)}{\mathbf{D}_V[\ell, \ell]} \left(\frac{\|\mathbf{Z}_{\mathcal{U}}[i]\|^2}{\mathbf{D}_{\mathcal{U}}[i,i]} - \frac{\mathbf{Z}_{\mathcal{U}}[i] \cdot \mathbf{Z}_{\mathcal{U}}[j]}{\sqrt{\mathbf{D}_{\mathcal{U}}[i,i]}\sqrt{\mathbf{D}_{\mathcal{U}}[j,j]}} \right)$$

$$= \sum_{u_i \in \mathcal{U}} \|\mathbf{Z}_{\mathcal{U}}[i]\|^2 - \sum_{v_\ell \in \mathcal{V}} \sum_{u_i, u_j \in \mathcal{U}} \mathbf{L}_{\mathcal{U}}[i, \ell] \cdot \mathbf{L}[j, \ell] \cdot \mathbf{Z}_{\mathcal{U}}[i] \cdot \mathbf{Z}_{\mathcal{U}}[j]$$

$$= \sum_{u_i} \mathbf{Z}_{\mathcal{U}}[i] \cdot \mathbf{Z}_{\mathcal{U}}[i] - \sum_{v_\ell \in \mathcal{V}} (\mathbf{L}_{\mathcal{U}}^{\top} \mathbf{Z}_{\mathcal{U}})[\ell] \cdot (\mathbf{L}_{\mathcal{U}}^{\top} \mathbf{Z}_{\mathcal{U}})[\ell]$$

$$= \text{trace}(\mathbf{Z}_{\mathcal{U}}^{\top} \cdot (\mathbf{I} - \mathbf{L}_{\mathcal{U}} \mathbf{L}_{\mathcal{U}}^{\top}) \cdot \mathbf{Z}_{\mathcal{U}}).$$

Next, we set the derivative of Eq. (24) w.r.t. $\mathbf{Z}_{\mathcal{U}}$ to zero and get the optimal $\mathbf{Z}_{\mathcal{U}}$ as:

$$\frac{\partial \{(1-\alpha) \cdot \|\mathbf{Z}_{\mathcal{U}} - \mathbf{X}_{\mathcal{U}}\|_F^2 + \alpha \cdot \text{trace}(\mathbf{Z}_{\mathcal{U}}^{\top}(\mathbf{I} - \mathbf{L}_{\mathcal{U}} \mathbf{L}_{\mathcal{U}}^{\top})\mathbf{Z}_{\mathcal{U}})\}}{\partial \mathbf{Z}_{\mathcal{U}}} = 0$$

$$\implies (1-\alpha) \cdot (\mathbf{Z}_{\mathcal{U}} - \mathbf{X}_{\mathcal{U}}) + \alpha(\mathbf{I} - \mathbf{L}_{\mathcal{U}} \mathbf{L}_{\mathcal{U}}^{\top})^{\top} \mathbf{Z}_{\mathcal{U}} = 0$$

$$\implies \mathbf{Z}_{\mathcal{U}} = ((1-\alpha) \cdot \mathbf{I} + \alpha \cdot (\mathbf{I} - \mathbf{L}_{\mathcal{U}} \mathbf{L}_{\mathcal{U}}^{\top}))^{-1} \cdot (1-\alpha)\mathbf{X}_{\mathcal{U}}.$$

By Neumann series, *i.e.*, $(\mathbf{I} - \mathbf{M})^{-1} = \sum_{k=0}^{\infty} \mathbf{M}^k$, we have

$$\left((1-\alpha) \cdot \mathbf{I} + \alpha(\mathbf{I} - \mathbf{L}_{\mathcal{U}} \mathbf{L}_{\mathcal{U}}^{\top}) \right)^{-1} = \left((1-\alpha) \cdot \mathbf{I} + \alpha(\mathbf{I} - \mathbf{L}_{\mathcal{U}} \mathbf{L}_{\mathcal{U}}^{\top}) \right)^{-1}$$

$$= \left(\mathbf{I} - \alpha \cdot \mathbf{L}_{\mathcal{U}} \mathbf{L}_{\mathcal{U}}^{\top} \right)^{-1} = \sum_{r=0}^{\infty} \alpha^r \cdot (\mathbf{L}_{\mathcal{U}} \mathbf{L}_{\mathcal{U}}^{\top})^r,$$

which seals the proof.   □

Proof of Lemma 3.1. We first need the following lemma:

LEMMA A.2. *The optimization objective in Eq. (8) is equivalent to optimizing:* $\max_{\mathbf{Y}} \text{trace}(\mathbf{Y}^{\top} \mathbf{S} \mathbf{Y})$, *where* $\mathbf{Y}$ *is defined in Eq. (10) and* $\mathbf{S}$ *is a* $|\mathcal{U}| \times |\mathcal{U}|$ *matrix where* $\mathbf{S}[i, j] = s(u_i, u_j)$.

Then, by the connection of the Frobenius norm and trace of matrices, *i.e.*, $\|\mathbf{M}\|_F^2 = \text{trace}(\mathbf{M}^{\top} \mathbf{M}) = \text{trace}(\mathbf{M}\mathbf{M}^{\top})$, we have

$$\mathcal{J} = \|\mathbf{R} - \mathbf{Y}\mathbf{H}^{\top}\|_F^2 = \text{trace}(\mathbf{R}\mathbf{R}^{\top} - \mathbf{R}\mathbf{H}\mathbf{Y}^{\top} - \mathbf{Y}\mathbf{H}^{\top}\mathbf{R}^{\top} + \mathbf{Y}\mathbf{H}^{\top}\mathbf{H}\mathbf{Y}^{\top})$$

$$= \text{trace}(\mathbf{R}\mathbf{R}^{\top}) - \text{trace}(\mathbf{R}\mathbf{H}\mathbf{Y}^{\top}) - \text{trace}(\mathbf{Y}\mathbf{H}^{\top}\mathbf{R}^{\top})$$

$$+ \text{trace}(\mathbf{Y}\mathbf{H}^{\top}\mathbf{H}\mathbf{Y}^{\top})$$

$$= \text{trace}(\mathbf{R}\mathbf{R}^{\top} - 2\mathbf{Y}\mathbf{H}^{\top}\mathbf{R}^{\top}) - \text{trace}(\mathbf{Y}^{\top}\mathbf{Y}\mathbf{H}^{\top}\mathbf{H})$$

$$= \text{trace}(\mathbf{R}\mathbf{R}^{\top} - 2\mathbf{Y}\mathbf{H}^{\top}\mathbf{R}^{\top} + \mathbf{H}^{\top}\mathbf{H}).$$

The zero gradient condition $\frac{\partial \mathcal{J}}{\partial \mathbf{H}} = -2\mathbf{R}^{\top}\mathbf{Y} + 2\mathbf{Y} = 0$ leads to $\mathbf{H} = \mathbf{R}^{\top}\mathbf{Y}$. Hence,

$$\mathcal{J} = \text{trace}(\mathbf{R}\mathbf{R}^{\top} - 2\mathbf{Y}\mathbf{H}^{\top}\mathbf{R}^{\top} + \mathbf{H}^{\top}\mathbf{H})$$

$$= \text{trace}(\mathbf{R}\mathbf{R}^{\top}) - 2 \text{trace}(\mathbf{Y}\mathbf{Y}^{\top}\mathbf{R}\mathbf{R}^{\top}) + \text{trace}(\mathbf{Y}^{\top}\mathbf{R}\mathbf{R}^{\top}\mathbf{Y})$$

$$= \text{trace}(\mathbf{R}\mathbf{R}^{\top}) - 2 \text{trace}(\mathbf{Y}^{\top}\mathbf{R}\mathbf{R}^{\top}\mathbf{Y}) + \text{trace}(\mathbf{Y}^{\top}\mathbf{R}\mathbf{R}^{\top}\mathbf{Y})$$

$$= \text{trace}(\mathbf{R}\mathbf{R}^{\top}) + \text{trace}(\mathbf{Y}^{\top}\mathbf{R}\mathbf{R}^{\top}\mathbf{Y}). \tag{25}$$

Since $\text{trace}(\mathbf{R}\mathbf{R}^{\top})$ is a constant, maximizing $\mathcal{J} = \|\mathbf{R} - \mathbf{Y}\mathbf{H}^{\top}\|_F^2$ is equivalent to maximizing $\text{trace}(\mathbf{Y}^{\top}\mathbf{R}\mathbf{R}^{\top}\mathbf{Y}) = \text{trace}(\mathbf{Y}^{\top}\mathbf{S}\mathbf{Y})$.   □

Proof of Lemma A.2. Let $\mathbf{S}$ be a $|\mathcal{U}| \times |\mathcal{U}|$ matrix where $\mathbf{S}[i, j] = s(u_i, u_j)$ and $\mathbf{S}_d$ be a $|\mathcal{U}| \times |\mathcal{U}|$ diagonal matrix in which $\mathbf{S}_d[i, i] = \sum_{u_j \in \mathcal{U}} s(u_i, u_j)$. Then,

$$\frac{1}{|C_\ell|} \sum_{u_i \in C_\ell, u_j \in \mathcal{U} \setminus C_\ell} s(u_i, u_j) = \frac{1}{2} \sum_{u_i, u_j \in \mathcal{U}} \mathbf{S}[i, j] \cdot (\mathbf{Y}[i, \ell] - \mathbf{Y}[j, \ell])^2$$

$$= \mathbf{Y}[:, \ell]^{\top} \cdot (\mathbf{S}_d - \mathbf{S}) \cdot \mathbf{Y}[:, \ell].$$

Thus, we can rewrite Eq. (8) as $\min_{\mathbf{Y}} \text{trace}(\mathbf{Y}^{\top}(\mathbf{S}_d - \mathbf{S})\mathbf{Y})$. Note that it can be further simplified as $\max_{\mathbf{Y}} \text{trace}(\mathbf{Y}^{\top}\mathbf{S}\mathbf{Y})$, which completes the proof.   □

Proof of Theorem 3.2. Let $\mathbf{z} = \widehat{\mathbf{Z}}_{\mathcal{U}}[i] - \widehat{\mathbf{Z}}_{\mathcal{U}}[j]$ and $\widehat{\mathbf{Q}} = \sqrt{d_{\mathcal{U}}} \cdot \mathbf{Q}$. Based on the definition of $\mathbf{R}'$ in Eq. (13), we can get

$$\mathbb{E}[\mathbf{R}'[i] \cdot \mathbf{R}'[j]] = \mathbb{E}\left[\frac{e}{d_{\mathcal{U}}} \sum_{\ell=1}^{d_{\mathcal{U}}} sin(\widehat{\mathbf{Q}}[\ell] \cdot \widehat{\mathbf{Z}}_{\mathcal{U}}[i]) \cdot sin(\widehat{\mathbf{Q}}[\ell] \cdot \widehat{\mathbf{Z}}_{\mathcal{U}}[j]) \right.$$

$$\left. - cos(\widehat{\mathbf{Q}}[\ell] \cdot \widehat{\mathbf{Z}}_{\mathcal{U}}[i]) \cdot cos(\widehat{\mathbf{Q}}[\ell] \cdot \widehat{\mathbf{Z}}_{\mathcal{U}}[j]) \right]$$

$$= \mathbb{E}\left[\frac{e}{d_{\mathcal{U}}} \sum_{\ell=1}^{d_{\mathcal{U}}} cos(\widehat{\mathbf{Q}}[\ell] \cdot (\widehat{\mathbf{Z}}_{\mathcal{U}}[i] - \widehat{\mathbf{Z}}_{\mathcal{U}}[j])) \right]$$

$$= \mathbb{E}\left[\frac{e}{d_{\mathcal{U}}} \sum_{\ell=1}^{d_{\mathcal{U}}} cos(\widehat{\mathbf{Q}}[\ell] \cdot \mathbf{z}) \right] \tag{26}$$

By Lemma 5 in [82], for any vector $\widehat{\mathbf{Q}}[\ell]$,

$$\left| \frac{\mathbb{E}[cos(\widehat{\mathbf{Q}}[\ell] \cdot \mathbf{z})]}{e^{-\|\mathbf{z}\|^2/2}} - \left(1 - \frac{\|\mathbf{z}\|^4}{4d_{\mathcal{U}}} \right) \right| \leq \frac{\|\mathbf{z}\|^4 \cdot (\|\mathbf{z}\|^4 + 8\|\mathbf{z}\|^2 + 8)}{16d_{\mathcal{U}}^2}.$$

Note that for each $u_i \in \mathcal{U}$, the row vector $\widehat{\mathbf{Z}}_{\mathcal{U}}[i]$ is L_2 normalized (Line 5), *i.e.*, $\|\mathbf{z}\|^2 \in [0, 1]$. Based thereon, the above inequality can be transformed into

$$1 - \frac{17}{16d_{\mathcal{U}}^2} - \frac{1}{4d_{\mathcal{U}}} \leq \frac{\mathbb{E}[cos(\widehat{\mathbf{Q}}[\ell] \cdot \mathbf{z})]}{e^{-\|\mathbf{z}\|^2/2}} \leq 1 + \frac{17}{16d_{\mathcal{U}}^2} + \frac{1}{4d_{\mathcal{U}}}. \tag{27}$$

According to Line 5 of Algorithm 1, $\|\widehat{\mathbf{Z}}_{\mathcal{U}}[i]\|^2 = 1 \, \forall u_i \in \mathcal{U}$. Thus,

$$\|\widehat{\mathbf{Z}}_{\mathcal{U}}[i] - \widehat{\mathbf{Z}}_{\mathcal{U}}[j]\|^2 = \|\widehat{\mathbf{Z}}_{\mathcal{U}}[i]\|^2 + \|\widehat{\mathbf{Z}}_{\mathcal{U}}[j]\|^2 - 2\widehat{\mathbf{Z}}_{\mathcal{U}}[i] \cdot \widehat{\mathbf{Z}}_{\mathcal{U}}[j]$$

$$= 2(1 - \widehat{\mathbf{Z}}_{\mathcal{U}}[i] \cdot \widehat{\mathbf{Z}}_{\mathcal{U}}[j]).$$

Thus, $e \cdot e^{-\frac{\|\mathbf{z}\|^2}{2}} = e \cdot e^{-\frac{\|\widehat{\mathbf{Z}}_{\mathcal{U}}[i] - \widehat{\mathbf{Z}}_{\mathcal{U}}[j]\|^2}{2}} = e^{\widehat{\mathbf{Z}}_{\mathcal{U}}[i] \cdot \widehat{\mathbf{Z}}_{\mathcal{U}}[j]}$. Combining Eq. (27) and Eq. (26) leads to

$$1 - \frac{17}{16d_{\mathcal{U}}^2} - \frac{1}{4d_{\mathcal{U}}} \leq \frac{\mathbb{E}[\mathbf{R}'[i] \cdot \mathbf{R}'[j]]}{e^{\widehat{\mathbf{Z}}_{\mathcal{U}}[i] \cdot \widehat{\mathbf{Z}}_{\mathcal{U}}[j]}} \leq 1 + \frac{17}{16d_{\mathcal{U}}^2} + \frac{1}{4d_{\mathcal{U}}}. \tag{28}$$

Further, Eq. (28) implies

$$1 - \frac{17}{16d_{\mathcal{U}}^2} - \frac{1}{4d_{\mathcal{U}}} \leq \frac{\mathbb{E}[\sum_{u_\ell \in \mathcal{U}} \mathbf{R}'[i] \cdot \mathbf{R}'[\ell]]}{\sum_{u_\ell \in \mathcal{U}} e^{\widehat{\mathbf{Z}}_{\mathcal{U}}[i] \cdot \widehat{\mathbf{Z}}_{\mathcal{U}}[\ell]}} \leq 1 + \frac{17}{16d_{\mathcal{U}}^2} + \frac{1}{4d_{\mathcal{U}}}. \tag{29}$$

According to Eq. (14), we can obtain

$$\mathbb{E}[\mathbf{R}[i] \cdot \mathbf{R}[j]] = \mathbb{E}\left[\frac{\mathbf{R}'[i] \cdot \mathbf{R}'[j]}{\sqrt{\mathbf{R}'[i] \cdot \mathbf{r}} \cdot \sqrt{\mathbf{R}'[j] \cdot \mathbf{r}}}\right].$$

Note that, by $\mathbf{r}$'s definition in Eq. (14), we have $\mathbb{E}[\mathbf{R}'[i] \cdot \mathbf{r}] = \mathbb{E}[\sum_{u_\ell \in \mathcal{U}} \mathbf{R}'[i] \cdot \mathbf{R}'[\ell]]$. Therefore, plugging Eq. (28) and Eq. (29) into the above equation proves the theorem. □

Proof of Lemma 3.4. According to Eq. (25), optimizing Eq. (9) is equivalent to maximizing $\text{trace}(\mathbf{Y}^\top \mathbf{R}\mathbf{R}^\top \mathbf{Y})$. Using the cyclic property of matrix trace, we have $\text{trace}(\mathbf{Y}^\top \mathbf{R}\mathbf{R}^\top \mathbf{Y}) = \text{trace}(\mathbf{Y}\mathbf{Y}^\top \mathbf{R}\mathbf{R}^\top)$ and $\text{trace}(\Upsilon^\top \mathbf{R}\mathbf{R}^\top \Upsilon) = \text{trace}(\Upsilon\Upsilon^\top \mathbf{R}\mathbf{R}^\top)$. Consequently,

$$\left| \|\mathbf{R} - \mathbf{Y}\mathbf{H}^\top\|_F^2 - \|\mathbf{R} - \Upsilon\mathbf{H}^\top\|_F^2 \right| \tag{30}$$
$$= \left| \text{trace}(\mathbf{Y}^\top \mathbf{R}\mathbf{R}^\top \mathbf{Y}) - \text{trace}(\Upsilon^\top \mathbf{R}\mathbf{R}^\top \Upsilon) \right|$$
$$= \left| \text{trace}((\mathbf{Y}\mathbf{Y}^\top - \Upsilon\Upsilon^\top)\mathbf{R}\mathbf{R}^\top) \right|.$$

The lemma is therefore proved. □

Proof of Lemma 3.3. First, define $\mathbf{P}_{\mathcal{U}}$ as follows:

$$\mathbf{P}_{\mathcal{U}} = (1 - \alpha) \sum_{r=0}^{\infty} \alpha^r \cdot \left(\mathbf{L}_{\mathcal{U}}\mathbf{L}_{\mathcal{U}}^\top\right)^r. \tag{31}$$

Suppose that $\Gamma\Sigma\Psi^\top$ be the exact top-d SVD of $\mathbf{X}_{\mathcal{U}}$, by Eckart–Young Theorem [15] (Theorem A.1), we have $\|\Gamma\Sigma\Psi^\top - \mathbf{X}_{\mathcal{U}}\|_2 \le \sigma_{d+1}$, where σ_{d+1} is the $(d+1)$-th largest singular value of $\mathbf{X}_{\mathcal{U}}$. Additionally, we can obtain $\|\Gamma\Sigma^2\Gamma^\top - \mathbf{X}_{\mathcal{U}}\mathbf{X}_{\mathcal{U}}^\top\|_2 \le \sigma_{d+1}^2$. By the definition of $\mathbf{P}_{\mathcal{U}}$ in Eq. (31), we have

$$\mathbf{P}_{\mathcal{U}} = \mathbf{D}_{\mathcal{U}}^{1/2} \cdot \sum_{r=0}^{\gamma} \alpha^t \cdot \left(\mathbf{D}_{\mathcal{U}}^{-1}\mathbf{B}\mathbf{D}_{\mathcal{V}}^{-1}\mathbf{B}^\top\right)^r \cdot \mathbf{D}_{\mathcal{U}}^{-1/2},$$

where $\mathbf{D}_{\mathcal{U}}^{-1}\mathbf{B}$ and $\mathbf{D}_{\mathcal{V}}^{-1}\mathbf{B}^\top$ are two row-stochastic matrices, i.e., the entries at each row sum up to 1. Let $\mathbf{M} = \mathbf{D}_{\mathcal{U}}^{-1}\mathbf{B}\mathbf{D}_{\mathcal{V}}^{-1}\mathbf{B}^\top$. Since the multiplication of two row-stochastic matrices yields a row-stochastic matrix, $\mathbf{M} = \mathbf{D}_{\mathcal{U}}^{-1}\mathbf{B}\mathbf{D}_{\mathcal{V}}^{-1}\mathbf{B}^\top$ is a row-stochastic matrix, which further connotes that $\mathbf{M}^r = \left(\mathbf{D}_{\mathcal{U}}^{-1}\mathbf{B}\mathbf{D}_{\mathcal{V}}^{-1}\mathbf{B}^\top\right)^r$ is row-stochastic, i.e., $\|\mathbf{M}[i]\|_1 = \sum_{u_\ell \in \mathcal{U}} \mathbf{M}[i, \ell] = 1 \ \forall u_i \in \mathcal{U}$. Hence, given a matrix $\Pi = \sum_{r=0}^{\infty} \alpha^r \mathbf{M}^r$,

$$\|\Pi[i]\|_1 = \sum_{r=0}^{\infty} \alpha^r = \frac{1}{1 - \alpha} \ \forall u_i \in \mathcal{U}.$$

Therefore, we can derive that

$$|\mathbf{F}[i] \cdot \mathbf{F}[j] - \mathbf{Z}_{\mathcal{U}}[i] \cdot \mathbf{Z}_{\mathcal{U}}[j]|$$
$$= |(\mathbf{P}_{\mathcal{U}}\Gamma\Sigma)[i] \cdot (\mathbf{P}_{\mathcal{U}}\Gamma\Sigma)[j] - (\mathbf{P}_{\mathcal{U}}\mathbf{X}_{\mathcal{U}})[i] \cdot (\mathbf{P}_{\mathcal{U}}\mathbf{X}_{\mathcal{U}})[j]|$$
$$= \left| \mathbf{P}_{\mathcal{U}}[i] \cdot (\Gamma\Sigma^2\Gamma^\top - \mathbf{X}_{\mathcal{U}}\mathbf{X}_{\mathcal{U}}^\top) \cdot \mathbf{P}_{\mathcal{U}}[j]^\top \right|$$
$$= \sum_{u_\ell \in \mathcal{U}} \mathbf{P}_{\mathcal{U}}[i, \ell] \cdot \sum_{u_h \in \mathcal{U}} \mathbf{P}_{\mathcal{U}}[j, h] \cdot \left(\Gamma\Sigma^2\Gamma^\top - \mathbf{X}_{\mathcal{U}}\mathbf{X}_{\mathcal{U}}^\top\right)[\ell, h]$$
$$\le \sum_{u_\ell \in \mathcal{U}} \mathbf{P}_{\mathcal{U}}[i, \ell] \cdot \sum_{u_h \in \mathcal{U}} \mathbf{P}_{\mathcal{U}}[j, h] \cdot \sigma_{d+1}^2$$
$$= \sum_{u_\ell \in \mathcal{U}} \sqrt{\frac{\mathbf{D}_{\mathcal{U}}[i, i]}{\mathbf{D}_{\mathcal{U}}[\ell, \ell]}} \cdot \Pi[i, \ell] \cdot \sum_{u_h \in \mathcal{U}} \sqrt{\frac{\mathbf{D}_{\mathcal{U}}[j, j]}{\mathbf{D}_{\mathcal{U}}[h, h]}} \cdot \Pi[j, h] \cdot \sigma_{d+1}^2$$
$$\le \sqrt{\mathbf{D}_{\mathcal{U}}[i, i] \cdot \mathbf{D}_{\mathcal{U}}[j, j]} \cdot \frac{\sigma_{d+1}^2}{1 - \alpha},$$

which finishes the proof. □

References

[1] Hasnat Ahmed, Yangyang Zhang, Muhammad Shoaib Zafar, Nasrullah Sheikh, and Zhenying Tai. 2020. Node embedding over attributed bipartite graphs. In *KSEM*. Springer, 202–210.
[2] Melissa Ailem, François Role, and Mohamed Nadif. 2015. Co-clustering document-term matrices by direct maximization of graph modularity. In *CIKM*. 1807–1810.
[3] Sajjad Athar, Rabeeh Ayaz Abbasi, Zafar Saeed, Anwar Said, Imran Razzak, and Flora D Salim. 2023. ASBiNE: Dynamic Bipartite Network Embedding for incorporating structural and attribute information. *WWW* (2023), 1–19.
[4] Cécile Bothorel, Juan David Cruz, Matteo Magnani, and Barbora Micenkova. 2015. Clustering attributed graphs: models, measures and methods. *Network Science* 3, 3 (2015), 408–444.
[5] Petr Chunaev. 2019. Community detection in node-attributed social networks: a survey. *arXiv preprint arXiv:1912.09816* (2019).
[6] Ganqu Cui, Jie Zhou, Cheng Yang, and Zhiyuan Liu. 2020. Adaptive graph encoder for attributed graph embedding. In *SIGKDD*. 976–985.
[7] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *TKDE* 31, 5 (2018), 833–852.
[8] Inderjit S Dhillon. 2001. Co-clustering documents and words using bipartite spectral graph partitioning. In *SIGKDD*. 269–274.
[9] Inderjit S Dhillon, Subramanyam Mallela, and Dharmendra S Modha. 2003. Information-theoretic co-clustering. In *SIGKDD*. 89–98.
[10] Chris Ding, Tao Li, Wei Peng, and Haesun Park. 2006. Orthogonal nonnegative matrix t-factorizations for clustering. In *SIGKDD*. 126–135.
[11] Issam Falih, Nistor Grozavu, Rushed Kanawati, and Younès Bennani. 2017. Anca: Attributed network clustering algorithm. In *Complex Networks*.
[12] Barakeel Fanseu Kamhoua, Lin Zhang, Kaili Ma, James Cheng, Bo Li, and Bo Han. 2023. Grace: A general graph convolution framework for attributed graph clustering. *TKDD* 17, 3 (2023), 1–31.
[13] Ming Gao, Leihui Chen, Xiangnan He, and Aoying Zhou. 2018. Bine: Bipartite network embedding. In *SIGIR*. 715–724.
[14] Edward Giamphy, Jean-Loup Guillaume, Antoine Doucet, and Kevin Sanchis. 2023. A survey on bipartite graphs embedding. *SNAM* 13, 1 (2023), 54.
[15] Gene H Gloub and Charles F Van Loan. 1996. Matrix computations. *Johns Hopkins Universtiy Press, 3rd edtion* (1996).
[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. 6.2. 2.3 softmax units for multinoulli output distributions. *Deep learning* 180 (2016).
[17] Gérard Govaert and Mohamed Nadif. 2013. *Co-clustering: models, algorithms and applications*. John Wiley & Sons.
[18] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. 855–864.
[19] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53, 2 (2011), 217–288.
[20] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *TIIS* 5, 4 (2015), 1–19.
[21] John A Hartigan and Manchek A Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *J R Stat Soc Ser C Appl Stat* 28, 1 (1979), 100–108.
[22] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*. 507–517.
[23] Wentao Huang, Yuchen Li, Yuan Fang, Ju Fan, and Hongxia Yang. 2020. Biane: Bipartite attributed network embedding. In *SIGIR*. 149–158.
[24] Xiao Huang, Jundong Li, and Xia Hu. 2017. Accelerated attributed network embedding. In *ICDM*. SIAM, 633–641.
[25] Junghoon Kim, Kaiyu Feng, Gao Cong, Diwen Zhu, Wenyuan Yu, and Chunyan Miao. 2022. ABC: attributed bipartite co-clustering. *PVLDB* 15, 10 (2022), 2134–2147.
[26] Jungeun Kim, Jae-Gil Lee, Byung Suk Lee, and Jiajun Liu. 2020. Geosocial co-clustering: A novel framework for geosocial community detection. *TIST* 11, 4 (2020), 1–26.
[27] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[28] Yuval Kluger, Ronen Basri, Joseph T Chang, and Mark Gerstein. 2003. Spectral biclustering of microarray data: coclustering genes and conditions. *Genome research* 13, 4 (2003), 703–716.
[29] Lazhar Labiod and Mohamed Nadif. 2011. Co-clustering for binary and categorical data with maximum modularity. In *ICDM*. IEEE, 1140–1145.
[30] Xinying Lai, Dingming Wu, Christian S Jensen, and Kezhong Lu. 2023. A Re-evaluation of Deep Learning Methods for Attributed Graph Clustering. In *CIKM*. 1168–1177.
[31] Andrea Lancichinetti, Santo Fortunato, and János Kertész. 2009. Detecting the overlapping and hierarchical community structure in complex networks. *New journal of physics* 11, 3 (2009), 033015.
[32] Daniel Lee and H Sebastian Seung. 2000. Algorithms for non-negative matrix factorization. *NeurIPS* 13 (2000).
[33] Yiran Li, Renchi Yang, and Jieming Shi. 2023. Efficient and Effective Attributed

Hypergraph Clustering via K-Nearest Neighbor Augmentation. *SIGMOD* 1, 2 (2023), 1–23.

[34] Zhao Li, Xin Shen, Yuhang Jiao, Xuming Pan, Pengcheng Zou, Xianling Meng, Chengwei Yao, and Jiajun Bu. 2020. Hierarchical bipartite graph neural networks: Towards large-scale e-commerce applications. In *ICDE*. 1677–1688.

[35] Jie Liu, Zhicheng He, Lai Wei, and Yalou Huang. 2018. Content to node: Self-translation network embedding. In *SIGKDD*. 1794–1802.

[36] Weifeng Liu, Jose C Principe, and Simon Haykin. 2011. *Kernel adaptive filtering: a comprehensive introduction*. Vol. 57. John Wiley & Sons.

[37] Yue Liu, Ke Liang, Jun Xia, Sihang Zhou, Xihong Yang, Xinwang Liu, and Stan Z. Li. 2023. Dink-Net: Neural Clustering on Large Graphs. (2023).

[38] Yao Ma, Xiaorui Liu, Tong Zhao, Yozen Liu, Jiliang Tang, and Neil Shah. 2021. A unified view on graph neural networks as graph signal denoising. In *CIKM*. 1202–1211.

[39] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[40] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).

[41] Glenn W Milligan and Martha C Cooper. 1985. An examination of procedures for determining the number of clusters in a data set. *Psychometrika* 50 (1985), 159–179.

[42] Nairouz Mrabah, Mohamed Bouguessa, Mohamed Fawzi Touati, and Riadh Ksantini. 2022. Rethinking graph auto-encoder models for attributed graph clustering. *TKDE* (2022).

[43] Robb J Muirhead. 2009. *Aspects of multivariate statistical theory*. John Wiley & Sons.

[44] Guosheng Pan, Yuan Yao, Hanghang Tong, Feng Xu, and Jian Lu. 2021. Unsupervised attributed network embedding via cross fusion. In *WSDM*. 797–805.

[45] Weisen Pan, Shizhan Chen, and Zhiyong Feng. 2013. Automatic clustering of social tag using community detection. *Applied Mathematics & Information Sciences* 7, 2 (2013), 675–681.

[46] Georgios A Pavlopoulos, Panagiota I Kontou, Athanasia Pavlopoulou, Costas Bouyioukos, Evripides Markou, and Pantelis G Bagos. 2018. Bipartite graphs in systems biology and medicine: a survey of methods and applications. *GigaScience* 7, 4 (2018), giy014.

[47] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. 701–710.

[48] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. 2019. Netsmf: Large-scale network embedding as sparse matrix factorization. In *WWW*. 1509–1520.

[49] Ali Rahimi and Benjamin Recht. 2007. Random features for large-scale kernel machines. *NeurIPS* 20 (2007).

[50] Yuxiang Ren, Hao Zhu, Jiawei Zhang, Peng Dai, and Liefeng Bo. 2021. Ensemfdet: An ensemble approach to fraud detection based on bipartite graph. In *ICDE*. 2039–2044.

[51] Yiye Ruan, David Fuhry, and Srinivasan Parthasarathy. 2013. Efficient community detection in large networks using content and links. In *WWW*.

[52] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *TPAMI* 22, 8 (2000), 888–905.

[53] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.

[54] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. Verse: Versatile graph embeddings from similarity measures. In *WWW*. 539–548.

[55] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2018. Deep Graph Infomax. In *ICLR*.

[56] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and computing* 17, 4 (2007), 395–416.

[57] Chun Wang, Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Attributed graph clustering: a deep attentional embedding approach. In *IJCAI*.

[58] Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. 2017. Mgae: Marginalized graph autoencoder for graph clustering. In *CIKM*.

[59] Chenguang Wang, Yangqiu Song, Ahmed El-Kishky, Dan Roth, Ming Zhang, and Jiawei Han. 2015. Incorporating world knowledge to document clustering via heterogeneous information networks. In *SIGKDD*. 1215–1224.

[60] Hao Wang, Enhong Chen, Qi Liu, Tong Xu, Dongfang Du, Wen Su, and Xiaopeng Zhang. 2018. A united approach to learning sparse attributed network embedding. In *ICDM*. IEEE, 557–566.

[61] David P Woodruff et al. 2014. Sketching as a tool for numerical linear algebra.

[62] *Foundations and Trends® in Theoretical Computer Science* 10, 1–2 (2014), 1–157.

[62] Tian Xie, Chaoyang He, Xiang Ren, Cyrus Shahabi, and C-C Jay Kuo. 2022. L-BGNN: Layerwise Trained Bipartite Graph Neural Networks. *TNNLS* (2022).

[63] Dongkuan Xu, Wei Cheng, Bo Zong, Jingchao Ni, Dongjin Song, Wenchao Yu, Yuncong Chen, Haifeng Chen, and Xiang Zhang. 2019. Deep co-clustering. In *SDM*. SIAM, 414–422.

[64] Panpan Xu, Nan Cao, Huamin Qu, and John Stasko. 2016. Interactive visual co-cluster analysis of bipartite graphs. In *PacificVis*. 32–39.

[65] Wei Xu, Xin Liu, and Yihong Gong. 2003. Document clustering based on non-negative matrix factorization. In *SIGIR*. 267–273.

[66] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. 2012. A model-based approach to attributed graph clustering. In *SIGMOD*.

[67] Zongyu Xu, Yihao Zhang, Long Yuan, Yuwen Qian, Zi Chen, Mingliang Zhou, Qin Mao, and Weibin Pan. 2023. Effective Community Search on Large Attributed Bipartite Graphs. *IJPRAI* 37, 02 (2023), 2359002.

[68] An Yan, Zhankui He, Jiacheng Li, Tianyang Zhang, and Julian McAuley. 2023. Personalized Showcases: Generating multi-modal explanations for recommendations. In *SIGIR*. 2251–2255.

[69] Hongqiang Yan, Yan Jiang, and Guannan Liu. 2018. Telecomm fraud detection via attributed bipartite network. In *ICSSSM*. 1–6.

[70] Liang Yang, Chuan Wang, Junhua Gu, Xiaochun Cao, and Bingxin Niu. 2021. Why do attributes propagate in graph convolutional neural networks?. In *AAAI*, Vol. 35. 4590–4598.

[71] Renchi Yang. 2022. Efficient and Effective Similarity Search over Bipartite Graphs. In *TheWebConf*. 308–318.

[72] Renchi Yang and Jieming Shi. 2024. Efficient High-Quality Clustering for Large Bipartite Graphs. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 23:1–23:27.

[73] Renchi Yang, Jieming Shi, Keke Huang, and Xiaokui Xiao. 2022. Scalable and Effective Bipartite Network Embedding. In *SIGMOD*. 1977–1991.

[74] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S Bhowmick. 2020. Homogeneous network embedding for massive graphs via reweighted personalized PageRank. *PVLDB* 13, 5 (2020), 670–683.

[75] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Sourav S Bhowmick, and Juncheng Liu. 2023. PANE: scalable and effective attributed network embedding. *VLDBJ* (2023), 1–26.

[76] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Juncheng Liu, and Sourav S Bhowmick. 2020. Scaling attributed network embedding to massive graphs. *PVLDB* 14, 1 (2020), 37–49.

[77] Renchi Yang, Jieming Shi, Yin Yang, Keke Huang, Shiqi Zhang, and Xiaokui Xiao. 2021. Effective and scalable clustering on massive attributed graphs. In *WWW*. 3675–3687.

[78] Renchi Yang, Yidu Wu, Xiaoyang Lin, Qichen Wang, Tsz Nam Chan, and Jieming Shi. 2024. Effective Clustering on Large Attributed Bipartite Graphs. *arXiv preprint arXiv:2405.11922* (2024).

[79] Tianbao Yang, Rong Jin, Yun Chi, and Shenghuo Zhu. 2003. Clustering relational data using attribute and link information. In *Proceedings of the text mining and link analysis workshop*. 9–15.

[80] Tianbao Yang, Rong Jin, Yun Chi, and Shenghuo Zhu. 2009. Combining Link and Content for Community Detection: A Discriminative Approach. In *SIGKDD*. 927–936.

[81] Ting Yao, Tao Mei, Chong-Wah Ngo, and Shipeng Li. 2013. Annotation for free: Video tagging by mining user search behavior. In *Multimedia*. 977–986.

[82] Felix Xinnan X Yu, Ananda Theertha Suresh, Krzysztof M Choromanski, Daniel N Holtmann-Rice, and Sanjiv Kumar. 2016. Orthogonal random features. *NeurIPS* 29 (2016).

[83] Hongyuan Zha, Xiaofeng He, Chris Ding, Horst Simon, and Ming Gu. 2001. Bipartite graph partitioning and data clustering. In *CIKM*. 25–32.

[84] Lili Zhang, Jennifer Priestley, Joseph DeMaio, Sherry Ni, and Xiaoguang Tian. 2021. Measuring customer similarity and identifying cross-selling products by community detection. *Big data* 9, 2 (2021), 132–143.

[85] Xiaotong Zhang, Han Liu, Qimai Li, and Xiao-Ming Wu. 2019. Attributed graph clustering via adaptive graph convolution. In *IJCAI*. 4327–4333.

[86] Yao Zhang, Yun Xiong, Xiangnan Kong, and Yangyong Zhu. 2017. Learning node embeddings in interaction graphs. In *CIKM*. 397–406.

[87] Tao Zhou, Jie Ren, Matúš Medo, and Yi-Cheng Zhang. 2007. Bipartite network projection and personal recommendation. *Physical review E* 76, 4 (2007), 046115.

[88] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2009. Graph Clustering Based on Structural/Attribute Similarities. *PVLDB* 2 (2009), 12.

[89] Meiqi Zhu, Xiao Wang, Chuan Shi, Houye Ji, and Peng Cui. 2021. Interpreting and unifying graph neural networks with an optimization framework. In *WWW*. 1215–1226.