Efficient and Effective Attributed Hypergraph Clustering via K-Nearest Neighbor Augmentation

YIRAN LI, Hong Kong Polytechnic University, China
RENCHI YANG*, Hong Kong Baptist University, China
JIEMING SHI†, Hong Kong Polytechnic University, China

Hypergraphs are an omnipresent data structure used to represent high-order interactions among entities. Given a hypergraph $\mathcal{H}$ wherein nodes are associated with attributes, *attributed hypergraph clustering* (AHC) aims to partition the nodes in $\mathcal{H}$ into k disjoint clusters, such that intra-cluster nodes are closely connected and share similar attributes, while inter-cluster nodes are far apart and dissimilar. It is highly challenging to capture multi-hop connections via nodes or attributes on large attributed hypergraphs for accurate clustering. Existing AHC solutions suffer from issues of prohibitive computational costs, sub-par clustering quality, or both. In this paper, we present AHCKA, an efficient approach to AHC, which achieves state-of-the-art result quality via several algorithmic designs. Under the hood, AHCKA includes three key components: (i) a carefully-crafted K-nearest neighbor augmentation strategy for the optimized exploitation of attribute information on hypergraphs, (ii) a joint hypergraph random walk model to devise an effective optimization objective towards AHC, and (iii) a highly efficient solver with speedup techniques for the problem optimization. Extensive experiments, comparing AHCKA against 15 baselines over 8 real attributed hypergraphs, reveal that AHCKA is superior to existing competitors in terms of clustering quality, while often being up to orders of magnitude faster.

CCS Concepts: • **Information systems** → **Clustering**; • **Computing methodologies** → *Cluster analysis*; • **Mathematics of computing** → *Computations on matrices*.

Additional Key Words and Phrases: Hypergraph, Clustering, Random Walk, Eigenvector

ACM Reference Format:
Yiran Li, Renchi Yang, and Jieming Shi. 2023. Efficient and Effective Attributed Hypergraph Clustering via K-Nearest Neighbor Augmentation. *Proc. ACM Manag. Data* 1, 2, Article 116 (June 2023), 23 pages. https://doi.org/10.1145/3589261

1 INTRODUCTION

Hypergraphs are a generalization of graphs wherein each edge can join an arbitrary number of nodes, referred to as a *hyperedge*. The hyperedge allows a more precise description of multilateral relationships between nodes in real-world graph data, such as collaboration relationships of multiple authors of a paper, interactions among proteins [8], products purchased together in one shopping cart, transactions involving multiple accounts [38]. In practice, nodes in hypergraphs are often associated with many attributes, e.g., the academic profile of authors and the descriptional data of

*Work partially done at the National University of Singapore.
†Corresponding Author.

Authors' addresses: Yiran Li, Hong Kong Polytechnic University, Hong Kong SAR, China, yi-ran.li@connect.polyu.hk; Renchi Yang, Hong Kong Baptist University, Hong Kong SAR, China, renchi@hkbu.edu.hk; Jieming Shi, Hong Kong Polytechnic University, Hong Kong SAR, China, jieming.shi@polyu.edu.hk.

2836-6573/2023/6-ART116 $15.00
https://doi.org/10.1145/3589261

products. The problem *attributed hypergraph clustering* (hereafter AHC) is to divide the n nodes in such an attributed hypergraph into k disjoint clusters such that nodes within the same cluster are close to each other with high connectedness and homogeneous attribute characteristics. AHC finds numerous real-life applications in community discovery [13], organization structure detection [6], Web query analysis [36], image classification [3] and segmentation [18], biological analysis [39], etc. As another example, AHC can cluster together academic publications with high research relevance by considering co-authorship hyperedges and keyword attributes in academic hypergraphs [7, 36].

Effective AHC computation is a highly challenging task, especially for large attributed hypergraphs with millions of nodes. First, nodes, hyperedge connections, and attributes are heterogeneous objects with inherently different traits, whose information cannot be seamlessly integrated in a simple and straightforward way. Second, as observed in previous works on simple graphs [42, 49], high-order relationships between nodes and node-attribute associations are crucial for clustering. However, modeling and computing such multi-hop relationships and associations via hyperedges usually with more than two nodes in attributed hypergraphs are rather difficult due to the complex hypergraph structures and prohibitive computational overheads (up to $O(n^2)$ in the worst case).

In the literature, a plethora of clustering solutions [11, 16, 20, 29, 47] are developed for plain hypergraphs. These methods overlook attribute information, leading to severely compromised AHC result quality. Besides, a large body of research on attributed graph clustering is conducted, resulting in a cornucopia of efficacious techniques [40, 42, 43, 46, 49]. However, most of these works are limited to small attributed graphs, and hence, cannot be directly applied to handle large attributed hypergraphs with more complex and unique structures. Inspired by the technical advances in the above fields, a number of efforts have been made towards AHC computation in the past years. The majority of AHC methods rely on non-negative matrix factorization [2, 6, 36], which requires numerous iterations of expensive matrix operations and even colossal space costs of materializing $n \times n$ dense matrices. Particularly, none of them take into account the high-order relationships between nodes, thereby limiting their result utility. The state-of-the-art approach GRAC [7] extends *graph convolution* [19] to hypergraphs, indirectly incorporating high-order relationships of nodes and attributes for clustering. Notwithstanding its enhanced clustering quality, GRAC runs in $O(n^2)$ time as an aftermath from costly graph convolution and SVD operations, which is prohibitive for large hypergraphs. To recapitulate, existing AHC approaches either yield sub-optimal clustering results or incur tremendous computational costs, rendering them impractical to cope with large attributed hypergraphs with millions of nodes.

Given the above, can we combine and orchestrate hypergraph topology and attribute information in an optimized way for improved clustering quality, while achieving high scalability over large attributed hypergraphs? This paper offers a positive answer by presenting AHCKA (Attributed Hypergraph Clustering via K-nearest neighbor Augmentation), a novel AHC approach that significantly advances the state of the art in AHC computation. AHCKA obtains superiority over existing solutions through several key techniques. The first one is a well-thought-out K-nearst neighbor (KNN) augmentation scheme, which augments the original hypergraph structure with a KNN graph containing additional connections constructed by adjacent nodes with K highest attribute similarities. This is inspired by a case study on a real dataset manifesting that incorporating all-pairwise node connections via attributes or none of them jeopardizes the empirical clustering quality. Second, AHCKA formulates the AHC task as a novel optimization problem based on a joint random walk model that allows for the seamless combination of high-order relationships from both the hypergraph and KNN graph. Further, AHCKA converts the original NP-hard problem into an approximate matrix trace optimization problem and harnesses efficient matrix operations to iteratively and greedily search for high-quality solutions. Lastly, AHCKA includes an effective initialization method that considerably facilitates the convergence of the optimization process using merely a handful of

iterations. We conduct extensive experiments on real-world attributed hypergraph data in different domains. Compared with existing baselines, AHCKA exhibits superior performance in terms of both clustering quality and efficiency. For instance, on the Amazon dataset with 2.27 million nodes, AHCKA gains over 10-fold speedup and a significant improvement of 4.8% in clustering accuracy, compared to the state of the art.

The contributions of this paper are summarized as follows:

- We devise a KNN augmentation scheme that exploits attributes to augment the original hypergraph structure in a cost-effective manner.
- We formulate the AHC task as an optimization problem with the objective of optimizing a quality measure based on a joint random walk model over the augmented hypergraph containing the original hypergraph and constructed KNN graph.
- We propose a number of techniques for efficient optimization of the objective, including a theoretically-grounded problem transformation, a greedy iterative framework, and an effective initialization approach that drastically reduces the number of iterations till convergence.
- The superior performance of AHCKA is validated by comprehensive experiments over real attributed hypergraph datasets.

2 PRELIMINARIES

Let $\mathcal{H} = \{\mathcal{V}, \mathcal{E}, \mathbf{X}\}$ be an attributed hypergraph, where $\mathcal{V}$ is the node set with cardinality $|\mathcal{V}| = n$, $\mathcal{E}$ is the hyperedge set with cardinality $|\mathcal{E}| = m$, and $\mathbf{X} \in \mathbb{R}^{n \times d}$ represents a node attribute matrix. Each hyperedge $e_i \in \mathcal{E}$ is a subset of $\mathcal{V}$ containing at least two nodes. A hyperedge e_i is said to be incident with a node v_j if $v_j \in e_i$. For each node $v_j \in \mathcal{V}$, its degree is denoted by $\delta(v_j)$, which is defined as the number of hyperedges incident to v_j. Each node v_j in $\mathcal{V}$ is associated with a d-dimensional attribute vector, denoted as $\mathbf{X}[j]$, i.e., the j-th row of the node attribute matrix $\mathbf{X}$. We denote by $\mathbf{A} \in \mathbb{R}^{m \times n}$ the incidence matrix of hypergraph $\mathcal{H}$, where each entry $\mathbf{A}[i, j] = 1$ if $v_j \in e_i$, otherwise $\mathbf{A}[i, j] = 0$. Let diagonal matrices $\mathbf{D}_V \in \mathbb{R}^{n \times n}$ and $\mathbf{D}_E \in \mathbb{R}^{m \times m}$ represent the degree matrix and hyperedge-size matrix of $\mathcal{H}$, where the diagonal entry $\mathbf{D}_V[j, j] = \delta(v_j)$ for $v_j \in \mathcal{V}$ and $\mathbf{D}_E[i, i] = |e_i|$ for $e_i \in \mathcal{E}$, respectively. The average degree of nodes in $\mathcal{H}$ is denoted as $\overline{\delta}$.

Attributed Hypergraph Clustering. Given an attributed hypergraph $\mathcal{H} = \{\mathcal{V}, \mathcal{E}, \mathbf{X}\}$ and a specified number k of clusters, the *attributed hypergraph clustering* (AHC) over $\mathcal{H}$ is to divide the node set $\mathcal{V}$ into k disjoint subsets $\{C_1, \ldots, C_k\}$ such that $\bigcup_{i=1}^{k} C_i = \mathcal{V}$ and the following properties are satisfied:

(1) Nodes within the same cluster are closely connected to each other in the hypergraph, while nodes in different clusters are far apart (**structure closeness**);
(2) Nodes in the same cluster have similar attribute values, while nodes in different clusters vary significantly in attribute values (**attribute homogeneity**).

Figure 1 exemplifies an attributed hypergraph $\mathcal{H}$ consisting of 8 nodes and 5 hyperedges, where each node is associated with an attribute vector and hyperedges e_1, e_2 contain 4 and 3 nodes, i.e., $\{v_1, v_2, v_4, v_5\}$ and $\{v_1, v_3, v_4\}$, respectively. $\mathcal{H}$ is partitioned into two clusters C_1 and C_2. We can observe that nodes v_1-v_5 in C_1 share similar attributes and are closely connected to each other, whereas nodes v_6, v_7 and v_8 form a cluster C_2 that is separated from C_1 with a paucity of connections and distinct attributes.

3 AHC AS AUGMENTED HYPERGRAPH CLUSTERING

According to the problem definition above, a central challenge is how to simultaneously exploit both hypergraph structure and attribute information for improved clustering quality. In literature, it

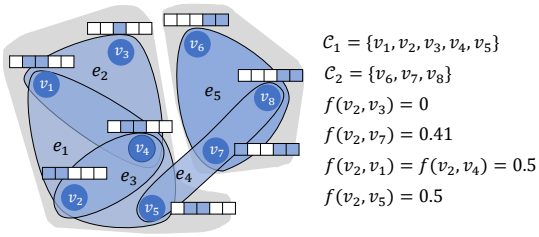


Fig. 1. An Example Attributed Hypergraph

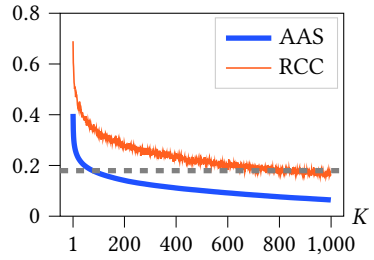


Fig. 2. AAS and RCC on Cora (best viewed in color)

is a natural and effective approach to augment network structure with attribute similarity strengths for simple attributed graph clustering [5, 42, 48, 49]. However, since a hypergraph yields different topological characteristics as illustrated in Figure 1, we argue that attribute augmentation should be conducted in a *controlable* way; otherwise, attributes may hamper, instead of improving, clustering quality, as shown in experiments (Section 7.3.1).

Therefore, in this section, we first develop a carefully-crafted augmentation strategy to augment attributes of nodes with hypergraph topology, which will benefit the clustering quality shown later on. As this attribute-based augmentation is orthogonal to the nature of hypergraph, its application to attributed graph clustering is also feasible, and we leave it to future works. Then we formulate Attributed Hypergraph Clustering as *Augmented Hypergraph Clustering*, with the same abbreviation AHC. Note that the augmented hypergraph involves both original hypergraph connections as well as augmented attribute connections. It is challenging to define a unified way to preserve the high-order information of both sides. To tackle this, we further design the (α, β, γ)-*random walk* to uniformly model the node relationships (in terms of both the structural closeness and attribute similarity) in the augmented hypergraphs. Based thereon, we define a multi-hop conductance (MHC), and formulate the objective of AHC as optimizing the conductance.

3.1 Augmented Hypergraph

Although the vanilla augmentation strategy improves the clustering quality in attributed graphs [5, 42, 48, 49], to our knowledge, its effectiveness over attributed hypergraphs is as of yet under-explored. Moreover, it requires constructing a densely connected graph, causing severe efficiency issues on large graphs. To this end, we first demystify the attribute homogeneity of nodes within the same cluster through an empirical study on a real-world attributed hypergraph, i.e., the Cora dataset[1] containing 2.7k academic papers in seven research fields (i.e., seven clusters). First, we use $f(v_i, v_j) = cosine(\mathbf{X}[i], \mathbf{X}[j])$ to denote the attribute similarity of nodes v_i, v_j. We refer to v_j as the K-th nearest neighbor of v_i if $f(v_i, v_j)$ is the K-th largest $\forall v_j \in \mathcal{V} \setminus v_i$. Figure 2 plots the *averaged attribute similarity* (AAS for short) $f(v_i, v_j)$ of any randomly picked node v_i and its K-th nearest neighbor v_j, and their ratio of co-occurring in the same cluster (RCC for short), when varying K from 1 to 1000. The AAS and RCC results from this real-world example demonstrate that two nodes with higher attribute similarity are also more likely to appear in the same cluster. Intuitively, applying the attribute-based augmentation strategy to hypergraphs can enhance the clustering results.

However, excessively augmenting the hypergraph with attribute information, namely, building too many connections between nodes according to attributes, will introduce distortion and adversely impact the clustering performance. To illustrate this, consider the example in Figure 1, where nodes

[1] https://people.cs.umass.edu/~mccallum/data.html

v_2, v_3 are in the same cluster as they share multiple common neighbors while v_2, v_7 are not. If we were to assign a cluster to node v_2 as per the additional connections created by attribute similarities, it is more likely to be v_2, v_7 rather than v_2, v_3 in the same cluster given $f(v_2, v_7) = 0.41 > f(v_2, v_3) = 0$, which is counter-intuitive.

Therefore, unlike the vanilla augmentation strategy employed in prior works, we propose a KNN augmentation strategy. That is, given the input attributed hypergraph $\mathcal{H} = \{\mathcal{V}, \mathcal{E}, \mathbf{X}\}$ and an integer K, we augment $\mathcal{H}$ with an undirected KNN graph $\mathcal{G}_K = \{\mathcal{V}, \mathcal{E}_K\}$. More specifically, for each node $v_i \in \mathcal{V}$, we identify K nodes in $\mathcal{V}$ (excluding v_i itself) that are most similar to v_i in terms of attribute similarity computed based on a similarity function $f(\cdot, \cdot)$ as v_i's neighbors in $\mathcal{G}_K$, denoted by $N_K(v_i)$. In other words, for every two nodes v_i, v_j ($v_j \in N_K(v_i)$), we construct an edge (v_i, v_j) with weight $f(\mathbf{X}[i], \mathbf{X}[j])$ in $\mathcal{E}_K$. Accordingly, the adjacency matrix $\mathbf{A}_K$ of $\mathcal{G}_K$ is defined as follows:

$$\mathbf{A}_K[i,j] = \begin{cases} 0, & \text{if } v_i \notin N_K(v_j) \text{ and } v_j \notin N_K(v_i), \\ 2 \cdot f(\mathbf{X}[i], \mathbf{X}[j]), & \text{if } v_i \in N_K(v_j) \text{ and } v_j \in N_K(v_i), \\ f(\mathbf{X}[i], \mathbf{X}[j]), & \text{otherwise.} \end{cases}$$

Thus, we obtain an augmented hypergraph $\mathcal{H}_A$ containing the hypergraph $\mathcal{H}_O = (\mathcal{V}, \mathcal{E})$ and the KNN graph $\mathcal{G}_K = (\mathcal{V}, \mathcal{E}_K)$. The reasons that we only consider K nearest neighbors for augmented hypergraph construction are three-fold. In the first place, the case study in Figure 2 suggests that there is no significant difference between the RCC of two random nodes (depicted by the gray dashed line) and that of two nodes v_i, v_j such that $v_j \in N_K(v_i)$, when K is beyond a number (roughly 500 in Figure 2). Therefore, such connections can be overlooked without impeding the clustering quality. Secondly, if we revisit the example in Figure 1 and apply the KNN strategy ($K = 3$) here, we can exclude the connection between v_2 and v_7 from $\mathcal{G}_K$ since $f(v_2, v_1) = f(v_2, v_4) = f(v_2, v_5) = 0.5 > f(v_2, v_7) = 0.41$. The distortion issue mentioned previously is therefore resolved. Furthermore, in comparison with the densely connected graph that encodes all attribute similarities (with up to $O(n^2)$ edges in the worst case), $\mathcal{G}_K$ can be efficiently constructed by utilizing well-established approximate nearest neighbor techniques in the literature, e.g., `ScaNN` [10].

The range of the KNN neighborhood is determined by parameter K. While a larger K allows the KNN graph to include more attribute similarity relations, this also leads to a higher proportion of unwanted inter-cluster edges in the KNN graph as evidenced by the lower RCC in Figure 2. Meanwhile, K cannot be too small (e.g., 5), or it will fail to utilize highly similar nodes that usually have high RCC. This trade-off of choosing K is experimentally studied in Section 7.3.1.

Now, the question lies in how to model the relationships of nodes in $\mathcal{V}$ of the augmented graph $\mathcal{H}_A$, which is a linchpin to AHC. In the following section, we present a joint random walk model that enables us to capture the multi-hop proximities of nodes over $\mathcal{H}_O$ and $\mathcal{G}_K$ jointly.

3.2 (α, β, γ)-Random Walk

Random walk with restart [34] (RWR) is one of the most common and effective random walk models for capturing the multi-hop relationships between nodes in a graph [15], and is widely used in many tasks such as ranking [34], recommendation [25], and clustering [1]. Given a graph $\mathcal{G}$, a source node u and a stopping probability α (typically $\alpha = 0.2$), at each step, an RWR originating from u either stops at the current node with probability α, or randomly picks an out-neighbor v of the current node according to the weight of edge (u, v) and navigates to v with the remaining $1 - \alpha$ probability. It follows that RWR score (a.k.a. *personalized PageRank* [14]) of any node pair (u, v) represents the probability that an RWR from u ends at node v. Intuitively, two nodes with dense (one-hop or multi-hop) connections should have a high RWR score.

Nevertheless, RWR is designed for general graphs, and thus cannot be directly applied to our augmented hypergraph $\mathcal{H}_A$ as it consists of a hypergraph $\mathcal{H}_O$ and a general graph $\mathcal{G}_K$. We devise a joint random walk scheme (i.e., (α, β, γ)-random walk), which conducts the RWR process over $\mathcal{H}_O$ and $\mathcal{G}_K$ jointly so as to seamlessly integrate topological proximity over both networks. Definition 1 states the formal definition of the (α, β, γ)-random walk process.

Definition 1. *Given an augmented hypergraph $\mathcal{H}_A = (\mathcal{H}_O, \mathcal{G}_K)$ and a source node u, an (α, β, γ)-random walk W starting from u conducts γ steps in total and at each step proceeds as follows.*

- *With probability α, W terminates at the current node v_i;*
- *with the other $1-\alpha$ probability, W navigates to a node v_j picked by the following rules:*
  - *with probability β_i, W draws an out-neighbor v_j of the current node v_i in $\mathcal{G}_K$ according to probability $\frac{\mathbf{A}_K[i,j]}{\sum_{v_l \in N_K(v_i)} \mathbf{A}_K[i,l]}$;*
  - *or with probability $1-\beta_i$, W first draws an hyperedge e_i incident to v_i in $\mathcal{H}_O$, and then draws node v_j from e_i uniformly at random.*

Notice that each node v_i is associated with a parameter β_i (see Eq. (1)) used to control the joint navigation between hypergraph $\mathcal{H}_O$ and KNN $\mathcal{G}_K$. The larger β_i is, the more likely that the random walk jumps to the neighbors of v_i in KNN $\mathcal{G}_K$.

$$\beta_i = \begin{cases} 0, & \text{if } \mathbf{X}[i] \text{ is a zero vector;} \\ 1, & \text{else if } \delta(v_i) = 0; \\ \beta, & \text{otherwise.} \end{cases} \tag{1}$$

In general, we set $\beta_i = \beta$, which is a user-specified parameter ranging from 0 to 1. In particular, when node v_i's attribute vector $\mathbf{X}[i]$ is a zero vector, i.e., v_i has no useful information in the KNN $\mathcal{G}_K$, we set β_i to 0. Conversely, β_i is configured as 1 if v_i is connected to none of the hyperedges, i.e., $\delta(v_i) = 0$. Let $s(v_i, v_j)$ denote the probability of an (α, β, γ)-random walk from v_i stopping at v_j in the end. Mathematically, based on Definition 1, we can derive the following formula for $s(v_i, v_j)$:

$$s(v_i, v_j) = \mathbf{S}[i, j] = \alpha \sum_{\ell=0}^{\gamma} (1-\alpha)^\ell \mathbf{P}^\ell[i, j], \tag{2}$$

where $\mathbf{P}$ is a transition matrix defined by

$$\mathbf{P} = (\mathbf{I} - \mathbf{B}) \cdot \mathbf{D}_V^{-1} \mathbf{A}^\mathsf{T} \mathbf{D}_E^{-1} \mathbf{A} + \mathbf{B} \mathbf{D}_K^{-1} \mathbf{A}_K, \tag{3}$$

$\mathbf{B} = diag(\beta_1, \ldots, \beta_n)$ is a diagonal matrix containing β_i parameters, and $\mathbf{D}_K$ is the diagonal degree matrix of $\mathcal{G}_K$. $\mathbf{P}^\ell[i, j]$ represents the probability that a ℓ-hop random walk from v_i terminates at v_j.

3.3 Objective Function

In what follows, we formally define the objective function of AHC. Intuitively, a high-quality cluster C in the augmented hypergraph $\mathcal{H}_A$ should be both internally cohesive and well disconnected from the remainder of the graph with the consideration of multi-hop connections. Hence, if we simulate an (α, β, γ)-random walk W from any node in C, W should have a low probability of escaping from C, i.e., ending at any node outside C. We refer to this escaping probability $\phi(C)$ as the *multi-hop conductance* (MHC) of C, which is defined in Eq. (4).

$$\phi(C) = \frac{1}{|C|} \sum_{v_i \in C} \sum_{v_j \notin C} s(v_i, v_j) \tag{4}$$

Since a low MHC $\phi(C)$ reflects a high coherence of cluster C, we then formulate AHC as an optimization problem of finding k clusters $\{C_1, \ldots, C_k\}$ such that their MHC $\Phi(\{C_1, \ldots, C_k\})$ (Eq. (5)) is minimized.

$$\Phi(\{C_1, \ldots, C_k\}) = \frac{1}{k} \sum_{C \in \{C_1, \ldots, C_k\}} \frac{1}{|C|} \sum_{v_i \in C} \sum_{v_j \notin C} s(v_i, v_j) \tag{5}$$

Directly minimizing Eq. (5) requires computing $s(v_i, v_j)$ (Eq. (2)) of every two nodes $v_i \in C, v_j \in \mathcal{V} \setminus C, \forall C \in \{C_1, C_2, \cdots, C_k\}$, which is prohibitively expensive for large graphs due to intractable computation time (i.e., $O(n^3)$) and storage space (i.e., $O(n^2)$). In addition, the minimization of $\Phi(\{C_1, \ldots, C_k\})$ is an NP-complete combinatorial optimization problem [30], rendering the exact solution unattainable on large graphs.

4 SOLUTION OVERVIEW

This section presents the top-level idea of our proposed solution, AHCKA, to AHC computation, and explains the intuitions behind it. At a high level, AHCKA first transforms the objective of AHC in Eq. (5) to a matrix trace maximization problem, and then derives an approximate solution via a top-k eigendecomposition.

Note that for any k non-overlapping clusters $\{C_1, C_2, \cdots, C_k\}$ on $\mathcal{H}$ satisfying $\bigcup_{i=1}^{k} C_i = \mathcal{V}$, they can be represented by a binary matrix $\mathbf{Y} \in \{0, 1\}^{n \times k}$, where for each node v_i and cluster C_j

$$\mathbf{Y}[i, j] = \begin{cases} 1, & v_i \in C_j \\ 0, & v_i \in \mathcal{V} \setminus C_j. \end{cases} \tag{6}$$

We refer to $\mathbf{Y}$ as a *binary cluster membership* (BCM) matrix of $\mathcal{H}$ and we use

$$h(\mathbf{Y}) = (\mathbf{Y}^\top \mathbf{Y})^{-1/2} \mathbf{Y} = \widehat{\mathbf{Y}} \tag{7}$$

to stand for the L_2 normalization of $\mathbf{Y}$. Particularly, $\widehat{\mathbf{Y}}$ has orthonormal columns, i.e., $\widehat{\mathbf{Y}}^\top \widehat{\mathbf{Y}} = \mathbf{I}_k$ where $\mathbf{I}_k$ is a $k \times k$ identity matrix. Given k non-overlapping clusters $\{C_1, C_2, \cdots, C_k\}$ and their corresponding BCM matrix $\mathbf{Y}$, it is trivial to show

$$\Phi(\{C_1, \ldots, C_k\}) = 1 - \Psi(\mathbf{Y}), \tag{8}$$

where $\Psi(\mathbf{Y})$ is defined as follows:

$$\Psi(\mathbf{Y}) = \frac{1}{k} trace(\widehat{\mathbf{Y}}^\top \mathbf{S} \widehat{\mathbf{Y}}). \tag{9}$$

Eq. (8) suggests that the minimization of MHC $\Phi(\{C_1, \ldots, C_k\})$ is equivalent to finding a BCM matrix $\mathbf{Y}$ such that the trace of matrix $\widehat{\mathbf{Y}}^\top \mathbf{S} \widehat{\mathbf{Y}}$ is maximized. Due to its NP-completeness, in lieu of computing the exact solution, we resort to utilizing a two-phase strategy to derive an approximate solution as follows.

If we relax the binary constraint on $\mathbf{Y}$, the following lemma establishes an upper bound ψ_σ for $\Psi(\mathbf{Y})$.

Lemma 4.1. *Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$ be the k largest singular values of matrix $\mathbf{S}$ in Eq. (2). Given any matrix $\mathbf{W} \in \mathbb{R}^{n \times k}$ such that $h(\mathbf{W})$ satisfies $h(\mathbf{W})^\top \cdot h(\mathbf{W}) = \mathbf{I}_K$, then $\Psi(\mathbf{W}) \leq \frac{1}{k} \sum_{i=1}^{k} \sigma_i = \psi_\sigma$.*

Lemma 4.1 implies that if we can first find a fractional matrix $\mathbf{W}$ such that $\Psi(\mathbf{W})$ is close to ψ_σ, a high-quality BCM matrix $\mathbf{Y}$ can be converted from $\mathbf{W}$ by leveraging algorithms such as k-Means [22]. Although we can obtain such a fractional matrix $\mathbf{W}$ by applying well-established trace maximization techniques [37] to Eq. (9), it still remains tenaciously challenging to compute $\mathbf{S}$. (All proofs are in Appendix.)

Lemma 4.2. *Let the columns of $\mathbf{Q} \in \mathbb{R}^{n \times k}$ be the second to $(k+1)$-th leading eigenvectors of $\mathbf{P}$ (Eq. (3)). Then, we have $\Psi(\mathbf{Q}) = \frac{1}{k} \sum_{i=2}^{k+1} \lambda_i = \psi_\lambda$, where $\lambda_2 \geq \cdots \geq \lambda_k \geq \lambda_{k+1}$ are the second to $(k+1)$-th leading eigenvalues of $\mathbf{S}$, sorted by algebraic value in descending order.*

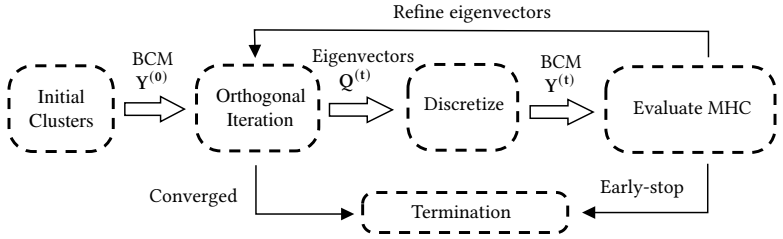


Fig. 3. Overview of AHCKA

By virtue of our analysis in Lemma 4.2, the second to $(k+1)$-th leading eigenvectors[2] $\mathbf{Q}$ of $\mathbf{P}$ (see Eq. (3)) can be regarded as a rough $\mathbf{W}$ since $\Psi(\mathbf{Q}) = \psi_\lambda \leq \psi_\sigma$ and the gap between ψ_λ and ψ_σ is insignificant in practice. For instance, on the Cora dataset, we can obtain $\psi_\sigma = 0.668$ and $\psi_\lambda = 0.596$, both of which are far superior to $\Psi(\mathbf{Y}^*) = 0.533$ of the ground-truth BCM matrix $\mathbf{Y}^*$. Consequently, using the second to $(k+1)$-th leading eigenvectors $\mathbf{Q}$ of $\mathbf{P}$ as the fractional solution $\mathbf{W}$ is sufficient to derive a favorable BCM matrix. Moreover, in doing so, we can avoid the tremendous overhead incurred by the materialization of $\mathbf{S}$.

To summarize, AHCKA adopts a two-phase strategy to obtain an approximate solution to the AHC problem. First, AHCKA computes the second to $(k+1)$-th leading eigenvectors $\mathbf{Q}$ of $\mathbf{P}$. After that, AHCKA transforms $\mathbf{Q}$ into a BCM matrix $\mathbf{Y}$ through a discretization approach [44] that minimizes the difference between $\mathbf{Q}$ and $\mathbf{Y}$. The rationale is that $\Psi(\mathbf{Q}) = \Psi(\mathbf{QR})$ if $\mathbf{R}$ is a $k \times k$ orthogonal matrix ensuring $\mathbf{R}^\mathsf{T}\mathbf{R} = \mathbf{I}_k$. Accordingly, we can derive a BCM matrix $\mathbf{Y} = \mathbf{QR}$ by minimizing the Frobenius norm $||\mathbf{Q} - \mathbf{QR}||_F$ with a binary constraint exerted on $\mathbf{QR}$. Note that we do not adopt k-Means over $\mathbf{Q}$ to get the BCM matrix $\mathbf{Y}$ as it deviates from our optimization objective in Eq. (9), and thus, produces sub-par result quality, as revealed by our experiments (Table 5).

Nevertheless, to realize the above idea, there still remain two crucial technical issues to be addressed:

(1) The brute-force computation of $\mathbf{Q}$ is highly time-consuming as it requires numerous iterations and the construction of $\mathbf{P}$.
(2) In practice, directly utilizing the exact or near-exact $\mathbf{Q}$ might incur overfitting towards the objective instead of ground-truth clusters, and hence, lead to sub-optimal clustering quality. It is challenging to derive a practically effective and robust BCM matrix $\mathbf{Y}$ from $\mathbf{Q}$.

5 THE AHCKA ALGORITHM

To circumvent the above challenges, AHCKA integrates the aforementioned two-phase scheme into an iterative framework, which enables us to approximate the second to $(k+1)$-th leading eigenvectors $\mathbf{Q}$ without constructing $\mathbf{P}$ explicitly, and greedily search the BCM matrix $\mathbf{Y}$ with the best MHC. Figure 3 sketches the main ingredients and algorithmic procedure of AHCKA. More specifically, AHCKA employs *orthogonal iterations* [28] to approximate the second to $(k+1)$-th leading eigenvectors $\mathbf{Q}$ of $\mathbf{P}$. During the course, AHCKA starts with an initial BCM matrix, followed by an orthogonal iteration to compute an approximate $\mathbf{Q}$ and an updated BCM matrix $\mathbf{Y}$ from the $\mathbf{Q}$ through `Discretize` algorithm [44]. Afterward, AHCKA inspects if $\mathbf{Q}$ reaches convergence and computes the MHC with the current BCM matrix $\mathbf{Y}$ via `CalMHC` algorithm (Algorithm 2). If $\mathbf{Q}$ converges (i.e., the BCM remains nearly stationary) or the early termination condition is satisfied

[2]We exclude the first eigenvector $\frac{1}{\sqrt{n}} \cdot \mathbf{1}$ of $\mathbf{P}$ as it is useless for clustering.

Algorithm 1: AHCKA

Input: Hypergraph $\mathcal{H}$, KNN transition matrix $\mathbf{P}_K$, the number of clusters k, diagonal matrix $\mathbf{B}$, constant α, error threshold ϵ_Q, the numbers of iterations T_a, γ, an integer τ, and an initial BCM matrix $\mathbf{Y}^{(0)}$.

Output: BCM matrix $\mathbf{Y}$

1 $\mathbf{Y} \leftarrow \mathbf{Y}^{(0)}$, $\widehat{\mathbf{Y}}^{(0)} \leftarrow h(\mathbf{Y}^{(0)})$;
2 $\mathbf{Q}^{(0)} \leftarrow \frac{1}{\sqrt{n}} \cdot \mathbf{1} | \widehat{\mathbf{Y}}^{(0)}$;
3 **for** $t \leftarrow 1, 2, \cdots, T_a$ **do**
4 Compute $\mathbf{Z}^{(t)}$ according to Eq. (11);
5 $\mathbf{Q}^{(t)}, \mathbf{R}^{(t)} \leftarrow \mathtt{QR}(\mathbf{Z}^{(t)})$;
6 **if** $t \ mod \ \tau = 0$ **then**
7 $\mathbf{Y}^{(t)} \leftarrow \mathtt{Discretize}(\mathbf{Q}^{(t)})$;
8 $\Phi(\mathbf{Y}^{(t)}) \leftarrow \mathtt{CalMHC}(\mathbf{Y}^{(t)}, \mathbf{P}_V, \mathbf{P}_E, \mathbf{P}_K, \mathbf{B}, \gamma, \alpha)$;
9 **if** $\Phi(\mathbf{Y}^{(t)}) < \Phi(\mathbf{Y})$ **then** $\mathbf{Y} \leftarrow \mathbf{Y}^{(t)}$;
10 **if** *Eq.* (14) *or Eq.* (15) *holds* **then break**;
11 **return** $\mathbf{Y}$;

(i.e., the MHC of current $\mathbf{Y}$ is satisfying), AHCKA terminates. Otherwise, AHCKA enters into the next orthogonal iteration with the updated $\mathbf{Q}$ and $\mathbf{Y}$.

In what follows, a detailed description of AHCKA is given in Section 5.1. Section 5.2 introduces an effective approach InitBCM for initializing the BCM matrix $\mathbf{Y}$, which drastically curtails the number of iterations needed and significantly boosts the computation efficiency of AHCKA. The complexity of the complete algorithm is analyzed in Section 5.3.

5.1 Main Algorithm

The pseudo-code of AHCKA is presented in Algorithm 1, which takes as input an attributed hypergraph $\mathcal{H}$, transition matrix of attribute KNN graph $\mathbf{P}_K$, the number k of clusters, a diagonal matrix $\mathbf{B}$ containing n parameters defined in Eq. (1), the random walk stopping probability α, an error threshold ϵ_Q, the numbers γ, T_a of iterations, an integer τ, and an initial BCM matrix $\mathbf{Y}^{(0)}$. AHCKA starts by computing the normalized BCM matrix $\widehat{\mathbf{Y}}^{(0)} = h(\mathbf{Y}^{(0)})$ (Eq. (7)) and setting the initial $k+1$ leading eigenvectors $\mathbf{Q}^{(0)}$ as $\frac{1}{\sqrt{n}} \cdot \mathbf{1} | \widehat{\mathbf{Y}}^{(0)}$ (Lines 1-2), where $|$ represents the horizontal concatenation and $\frac{1}{\sqrt{n}} \cdot \mathbf{1}$ is the first leading eigenvector of $\mathbf{P}$ since it is a stochastic matrix. After that, AHCKA enters into at most T_a orthogonal iterations for computing the $k+1$ leading eigenvectors $\mathbf{Q}$ and the BCM matrix $\mathbf{Y}$ (Lines 3-10).

At step t, orthogonal iteration essentially updates the approximate $k+1$ leading eigenvectors of $\mathbf{P}$ as $\mathbf{Q}^{(t)}$ according to the formula below (Lines 4-5):

$$\mathbf{Q}^{(t)}\mathbf{R}^{(t)} = \mathbf{Z}^{(t)} = \mathbf{P}\mathbf{Q}^{(t-1)}, \tag{10}$$

where $\mathbf{Q}^{(t)}$ is obtained by a QR decomposition over $\mathbf{Z}^{(t)}$. If t is sufficiently large, $\mathbf{Q}^{(t)}$ will converge to the exact $k+1$ leading eigenvectors of $\mathbf{P}$ [28]. Note that the direct computation of $\mathbf{Z}^{(t)} = \mathbf{P}\mathbf{Q}^{t-1}$ requires constructing $\mathbf{P}$ explicitly as per Eq. (3), which incurs an exorbitant amount of time and space (up to $O(n^2)$ in the worst case). To mitigate this issue, we decouple and reorder the matrix

Algorithm 2: CalMHC

Input: $\mathbf{Y}^{(t)}, \mathbf{P}_V, \mathbf{P}_E, \mathbf{P}_K, \mathbf{B}, \gamma, \alpha$
Output: MHC ϕ_t
1 $\widehat{\mathbf{Y}}^{(t)} \leftarrow h(\mathbf{Y}^{(t)})$; $\mathbf{H}^{(0)} \leftarrow \alpha\widehat{\mathbf{Y}}^{(t)}$;
2 **for** $\ell \leftarrow 1, 2, \ldots \gamma$ **do**
3 Compute $\mathbf{H}^{(\ell)}$ according to Eq. (13);
4 $\phi_t \leftarrow 1 - \frac{1}{k} trace(\widehat{\mathbf{Y}}^{(t)\top}\mathbf{H}^{(\gamma)})$;
5 **return** ϕ_t ;

multiplication as in Eq. (11)

$$\mathbf{Z}^{(t)} = (\mathbf{I} - \mathbf{B}) \cdot \mathbf{P}_V \cdot \left(\mathbf{P}_E\mathbf{Q}^{(t-1)}\right) + \mathbf{B}\mathbf{P}_K \cdot \mathbf{Q}^{(t-1)}, \tag{11}$$

where

$$\mathbf{P}_V = \mathbf{D}_V^{-1}\mathbf{A}^{\mathsf{T}},\ \mathbf{P}_E = \mathbf{D}_E^{-1}\mathbf{A}, \tag{12}$$

are two sparse matrices of $\mathcal{H}$ and $\mathbf{P}_K = \mathbf{D}_K^{-1}\mathbf{A}_K$ is the sparse transition matrix of the KNN graph $\mathcal{G}_K$ defined in Section 3.1. Note that all of them can be efficiently constructed in the preprocessing stage. As such, we eliminate the need to materialize $\mathbf{P}$ and reduce the time complexity of computing $\mathbf{Z}^{(t)}$ to $O(nk \cdot (\overline{\delta} + K))$.

After obtaining $\mathbf{Q}^{(t)}$, AHCKA converts $\mathbf{Q}^{(t)}$ into a new BCM matrix $\mathbf{Y}^{(t)}$ (Lines 6-7) using the Discretize algorithm [44]. Notice that we conduct this conversion every other τ iterations in order to avert unnecessary operations as the difference between $\mathbf{Y}^{(t)}$ and $\mathbf{Y}^{(t-1)}$ is often insignificant.

Next, at Line 8, AHCKA invokes CalMHC (i.e., Algorithm 2) with a BCM matrix $\mathbf{Y}^{(t)}$, other parameters including $\mathbf{P}_V, \mathbf{P}_E, \mathbf{P}_K, \mathbf{B}, \alpha$, and the number of iterations γ as input to calculate the MHC ϕ_t of the current BCM matrix $\mathbf{Y}^{(t)}$. To avoid the materialization of $\mathbf{S}$ required in Eq. (8) and Eq. (9), Algorithm 2 computes ϕ_t in an iterative manner by reordering the matrix multiplications (Lines 2-3 in Algorithm 2). More precisely, at the ℓ-th iteration, it obtains the intermediate result $\mathbf{H}^{(\ell)}$ via the following equation:

$$\mathbf{H}^{(\ell)} = (1-\alpha)\left((\mathbf{I} - \mathbf{B}) \cdot \mathbf{P}_V \cdot \left(\mathbf{P}_E\mathbf{H}^{(\ell-1)}\right) + \mathbf{B}\mathbf{P}_K \cdot \mathbf{H}^{(\ell-1)}\right) + \mathbf{H}^{(0)}. \tag{13}$$

$\mathbf{H}^{(0)}$ is initialized as Line 1 in Algorithm 2. It can be verified that MHC $\phi_t = 1 - \frac{1}{k} trace(\widehat{\mathbf{Y}}^{(t)\top}\mathbf{H}^{(\gamma)})$ (Line 4 in Algorithm 2).

$$||\mathbf{Q}^{(t)} - \mathbf{Q}^{(t-1)}|| < \epsilon_Q \tag{14}$$

Once the convergence criterion of $\mathbf{Q}^{(t)}$ (Eq. (14)) is satisfied, or the early termination condition (Eq. (15)) holds, AHCKA ceases the iterative process and returns the BCM matrix $\mathbf{Y}$ with the lowest MHC value (Lines 9-11 in Algorithm 1).

$$\phi_{t-2\tau} < \phi_{t-\tau} < \phi_t \tag{15}$$

Otherwise, AHCKA proceeds to the next orthogonal iteration. The rationale for the early termination condition in Eq. (15) is that after t iterations of successive increments in MHC, ϕ_t attains a near-optimal value in practice.

5.2 Greedy Initialization of BCM

Akin to many optimization problems, AHCKA requires a large number of iterations to achieve convergence when $\mathbf{Y}^{(0)}$ is randomly initialized. To tackle this issue, in this section, we propose a

Algorithm 3: `InitBCM`

Input: Hyergraph $\mathcal{H}$, matrices $\mathbf{P}_V, \mathbf{P}_E$, integer k, constant α, the number of iterations T_i.
Output: An initial BCM matrix $\mathbf{Y}^{(0)}$.

1 $\mathcal{V}_c \leftarrow$ The sorted indices of nodes with k largest degrees;
2 Initialize $\mathbf{Z}_0 \leftarrow \mathbf{0}^{k\times n}$;
3 **for** $j \leftarrow 1$ *to* k **do** $\mathbf{Z}_0[j, \mathcal{V}_c(j)] \leftarrow 1$;
4 Initialize $\mathbf{\Pi}_c^{(0)} \leftarrow \alpha\mathbf{Z}_0$;
5 **for** $t \leftarrow 1, 2, \ldots T_i$ **do**
6 Compute $\mathbf{\Pi}_c^{(t)}$ according to Eq. (16);
7 **for** $v_j \in \mathcal{V}$ **do**
8 Calculate $l(v_j)$ according to Eq. (17);
9 $\mathbf{Y}^{(0)}[j, l(v_j)] \leftarrow 1$;
10 **return** $\mathbf{Y}^{(0)}$;

greedy initialization technique, `InitBCM`, whereby we can immediately gain a passable BCM matrix $\mathbf{Y}^{(0)}$ and expedite the convergence, as demonstrated by our experiments in Section 7.4.

The rationale of `InitBCM` is inspired by the intuition that most nodes tend to cluster together around a number of center nodes [26, 32]. Therefore, we can first pick a set $\mathcal{V}_c$ of top influential nodes with respect to the whole hypergraph, and calculate the multi-hop proximities (i.e., RWR scores) of each node to the influential nodes $\mathcal{V}_c$ (i.e., centers). Then, the cluster center of each node can be determined by its proximity to nodes in $\mathcal{V}_c$ accordingly.

Algorithm 3 displays the pseudo-code of `InitBCM`. Given hypergraph $\mathcal{H}$, and transition matrices $\mathbf{P}_V, \mathbf{P}_E$ defined in Eq. (12), the number k of clusters, random walk stopping probability α, and the number of iterations T_i, as input, `InitBCM` begins by initializing an ordered set $\mathcal{V}_c$ consisting of the k nodes with k largest degrees in $\mathcal{H}$ (sorted by their indices), which later serves as the cluster centers (Line 1). Then, a $k \times n$ matrix $\mathbf{Z}_0$ is created, where for each integer $j \in [1, k]$, $\mathbf{Z}_0[j, \mathcal{V}_c(j)]$ is set to 1 and 0 otherwise and $\mathcal{V}_c(j)$ denotes the node index of the j-th node in $\mathcal{V}_c$ (Lines 2-3). Next, `InitBCM` launches T_i iterations to calculate the RWR scores of all nodes w.r.t the k nodes in $\mathcal{V}_c$ (Lines 4-5). More precisely, at t-th iteration, we compute approximate RWR scores $\mathbf{\Pi}_c^{(t)}$ as follows (Line 6):

$$\mathbf{\Pi}_c^{(t)} = (1-\alpha)\left(\mathbf{\Pi}_c^{(t-1)}\mathbf{P}_V\right) \cdot \mathbf{P}_E + \mathbf{\Pi}_0, \tag{16}$$

where $\mathbf{\Pi}_0 = \alpha\mathbf{Z}_0$ (Line 5). Note that we reorder the matrix multiplications as in Eq. (16) so as to bypass the materialization of the $n \times n$ matrix $\mathbf{P}_V\mathbf{P}_E$. After obtaining $\mathbf{\Pi}_c^{(T_i)}$, `InitBCM` assigns the node $\mathcal{V}_c(l(v_j))$ as the cluster center to each node v_j in $\mathcal{H}$ as per Eq. (17) (Lines 7-9).

$$l(v_j) = \arg\max_{1\le l\le k} \mathbf{\Pi}_c^{(t)}[l, j], \tag{17}$$

meaning that we pick a cluster center from $\mathcal{V}_c$ such that its RWR score $\mathbf{\Pi}_c^{(t)}[l, j]$ w.r.t node v_j is the highest. Finally, an $n \times k$ binary matrix $\mathbf{Y}^{(0)}$ is constructed by setting $\mathbf{Y}^{(0)}[j, l(v_j)]$ to 1 for $v_j \in \mathcal{V}$ and returned as the initial BCM matrix.

5.3 Analysis

One of the main computational costs of `AHCKA` stems from the sparse matrix multiplications, i.e., Line 4 in Algorithm 1, Line 3 in Algorithm 2, and Line 6 in Algorithm 3. We first consider Line 4 in Algorithm 1, i.e., Eq. (11). Since $\mathbf{Q}^{(t-1)}$ is an $n \times (k+1)$ matrix and the numbers of non-zero entries

in sparse matrices $\mathbf{P}_V$, $\mathbf{P}_E$, and $\mathbf{P}_K$ are $n\overline{\delta}$, $n\overline{\delta}$, and nK, respectively, its complexity is $O((n\overline{\delta}+nK)\cdot k)$ [45]. Analogously, according to Eq. (13), and Eq. (16), both the time costs of Line 3 in Algorithm 2 and Line 6 in Algorithm 3 are bounded by $O(n\overline{\delta}k)$. Recall that these three operations are conducted up to T_a, γ, and T_i times in Algorithms 1, 2, and 3, respectively. Therefore, the total time cost of sparse matrix multiplications is $O(kn\overline{\delta}\cdot(T_a+T_i+\gamma)+knKT_a)$. Moreover, in Algorithm 1, the QR decomposition at Line 5 takes $O(k^2n)$ time and the Discretize algorithm [44] runs in $O(k^2n+k^3)$ time. Overall, the time complexity of AHCKA is $O(kn\overline{\delta}\cdot(T_a+T_i+\gamma)+knKT_a+k^2n)$, which equals $O(n\overline{\delta})$ when T_a, T_i, γ, k, and K are regarded as constants. The space complexity of AHCKA is $O(n\cdot(\overline{\delta}+K+k))$ as all matrices are in sparse form.

6 RELATED WORK

Hypergraph Clustering. Motivated by the applications in circuit manufacturing, partitioning algorithms have been developed to divide hypergraphs into partitions/clusters, such as hMetis [16] and KaHyPar [29]. These methods typically adopt a three-stage framework consisting of coarsening, initial clustering, and refinement stages. These algorithms directly perform clustering on a coarsened hypergraph with a relatively small size. In addition, they run a portfolio of clustering algorithms and select the best outcome. These algorithms rely on a set of clustering heuristics and lack the extensibility for exploiting node attribute information.

Hypergraph Normalized Cut (HNCut) [47] is a conductance measure for hypergraph clusters from which the normalized hypergraph Laplacian $\mathbf{\Delta}=\mathbf{I}-\mathbf{\Theta}$ is derived for spectral clustering.

$$\mathbf{\Theta}=\mathbf{D}_V^{-1/2}\mathbf{A}^T\mathbf{D}_E^{-1}\mathbf{A}\mathbf{D}_V^{-1/2} \tag{18}$$

Alternatively, hGraclus [36] optimizes the HNCut objective using a multi-level kernel K-means algorithm. Non-negative matrix factorization has also been applied to hypergraph clustering [11]. Despite the theoretical soundness, these algorithms are less efficient than the aforementioned partitioning algorithms and they do not utilize node attributes either.

For the problem of hypergraph local clustering, which is to find a high-quality cluster containing a specified node, a sweep cut method is proposed [33] to find the cluster based on hypergraph Personalized PageRank (PPR) values. Their formulation of PPR is based on a non-linear hypergraph Laplacian operator [21] which can incur significant computation overhead for the global clustering algorithm. In this paper, we focus on global clustering, which is a different problem setting from local clustering.

Attributed Hypergraph Clustering. There exist studies designing dedicated clustering algorithms on attributed hypergraphs. JNMF [6] is an AHC algorithm based on non-negative matrix factorization (NMF). With normalized hypergraph Laplacian [47] matrix $\mathbf{\Delta}=\mathbf{I}-\mathbf{\Theta}$ and attribute matrix $\mathbf{X}$, JNMF optimizes the following joint objective that includes a basic NMF part as well as a symmetric NMF part.

$$\min_{\mathbf{W},\mathbf{H},\tilde{\mathbf{H}}\geq 0}||\mathbf{X}-\mathbf{W}\mathbf{H}||_F^2+\alpha||\mathbf{\Theta}-\tilde{\mathbf{H}}^{\mathrm{T}}\mathbf{H}||_F^2+\beta||\tilde{\mathbf{H}}-\mathbf{H}||_F^2 \tag{19}$$

With optimization using block coordinate descent (BCD) scheme, the matrix $\mathbf{H}$ is expected to encode cluster memberships. MEGA [36] extends the formulation of JNMF clustering objective for *semi-supervised* clustering of multi-view data containing hypergraph, node attributes as well as pair-wise similarity graph. MEGA's clustering performance is further enhanced by initialization with hGraclus algorithm.

GNMF [2] algorithm is originally proposed for high-dimensional data clustering, while the authors of [7] extend its objective with 3 different hypergraph structural encodings so that it spawns baseline methods for AHC. Among the 3 variants, GNMF-C directly builds upon the clique-expansion graph

with adjacency matrix $\mathbf{A}^\mathsf{T}\mathbf{A}$, GNMF-A uses the adjacency matrix $\mathbf{A}^\mathsf{T}\mathbf{A}-\mathbf{D}_V$ [27] which removes the self-loops, and GNMF-L employs matrix $\mathbf{\Theta}$ from hypergraph normalized Laplacian [47]. Although NMF-based algorithms sometimes produce clusters with good quality, their scalability is underwhelming as shown in our experiments.

As the state-of-the-art algorithm for attributed hypergraph clustering, GRAC [7] performs hypergraph convolution [41] on node attributes, which resembles the hypergraph diffusion process with mediators [4]. Clusters are predicted from the propagated features via a spectral algorithm and the authors proposed a stopping criterion based on intermediate predictions. We experimentally prove that AHCKA can outperform GRAC.

Attributed Graph Clustering. There exist a collection of studies on attributed graph clustering. These methods are designed without considering the hypergraph structure investigated in this paper. Some studies perform attributed graph clustering by adopting probabilistic models to combine graph structure with attributes, including discriminative models such as PCL-DC [43] and generative models such as BAGC [40]. Nevertheless, these methods are typically limited to handling categorical attributes. Moreover, inference over the probability distribution of $O(2^n)$ hyperedges poses a significant challenge against their generalization to hypergraph.

Within the random walk framework, SA-Cluster [49] algorithm augments the original graph with virtual nodes representing each possible attribute-value pair and performs k-Medroids clustering using a random walk distance measure. ACMin [42] defines attributed random walk by adding virtual attribute nodes as bridges and combines it with graph random walk into a joint transition matrix. In a fashion similar to GCN [19], AGC [46] performs graph convolution on node attributes to produce smooth feature representations that incorporate network structure information and subsequently applies spectral clustering. For their spectral algorithm, the authors also design heuristics to prevent propagated features from over-smoothing that undermines cluster quality.

As a workaround, it is possible to first apply graph reduction, *e.g.*, clique expansion or star expansion [35] on hypergraphs to convert them to simple graphs, and then apply attributed graph clustering methods (*e.g.*, ACMin) to get clustering results. However, the graph reduction will cause potential loss of the hypergraph structure, resulting in inferior performance to solve the AHC problem, which is validated in experiments. Therefore, to effectively produce high-quality clustering results for the AHC problem, one method should directly work on the hypergraph topology with attributes.

7 EXPERIMENTS

We experimentally evaluate the proposed AHCKA against 15 competitors in terms of both clustering quality and efficiency on 8 real-world attributed hypergraphs. All the experiments are conducted on a Linux machine powered by two Intel Xeon(R) Gold 6226R CPUs and 384GB RAM, using at most 16 threads. The implementation is available at https://github.com/CyanideCentral/AHCKA.

7.1 Experimental Setup

7.1.1 **Datasets.** In Table 1, we list the statistics of eight real-world attributed hypergraphs used in our experiments. $|\mathcal{V}|$ and $|\mathcal{E}|$ denote the number of nodes and hyperedges, respectively. We use $\overline{\delta}$ to signify the average node degree, d to represent the feature dimension, and k to denote the number of ground-truth clusters.

Query dataset [36] is a real-world Web query hypergraph, where nodes represent queries, and are connected by hyperedges representing query sessions, and nodes are associated with attributes extracted from webpage contents. Cora-CA, Cora-CC, Citeseer, and DBLP are four benchmark datasets used in prior work [41]. All of them are originally collected from academic databases

Table 1. Datasets statistics

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\overline{\delta}$	d	k
Query	481	15,762	88.49	426	6
Cora-CA	2,708	1,072	1.69	1,433	7
Cora-CC	2,708	1,579	1.77	1,433	7
Citeseer	3,312	1,079	1.04	3,703	6
20News	16,242	100	4.03	100	4
DBLP	41,302	22,363	2.41	1,425	6
Amazon	2,268,083	4,285,295	32.2	1,000	15
MAG-PM	2,353,996	1,082,711	7.34	1,000	22

(i.e., Cora, DBLP, and Citeseer), where each node represents a publication, node attributes are the unique words of the publication, and research topics are regarded as ground-truth clusters. Hyperedges correspond to co-authorship in Cora-CA and DBLP datasets or co-citation relationship in Cora-CC and Citeseer datasets. 20News dataset [12] consists of messages taken from Usenet [23] newsgroups. Messages are nodes, and the messages containing the same keyword are connected by a corresponding hyperedge, and the TF-IDF vector for each message is used as the node attributes. Amazon dataset is constructed based on the 5-core subset of Amazon reviews dataset [24], where each node represents a product and a hyperedge contains the products reviewed by a user. For each product, we use the associated textual metadata as the node attributes and the product category as its cluster label. MAG-PM dataset is extracted from the Microsoft Academic Graph [31], where nodes, co-authorship hyperedges, attributes, and cluster labels are obtained as in other academic datasets (i.e., Cora-CA, Cora-CC, Citeseer, and DBLP). Both Amazon and MAG-PM will be made publicly available.

7.1.2 **Competitors and Parameter Settings.** The competitors are summarized as follows:

- 3 plain hypergraph clustering methods including `HNCut` [47], `HyperAdj` [27], and `KaHyPar` [9];
- the extended AHC versions of the 3 methods above (dubbed as `ATHNCut`, `ATHyperAdj`, and `ATKaHyPar`, respectively), which work on an augmented hypergraph with attribute-KNN hyperedges of all nodes merged into the input hypergraph;
- `ATMetis` that applies the traditional graph clustering algorithm `Metis` [17] over a graph constructed by clique expansion of the input hypergraph and attribute KNN graph augmentation;
- 5 AHC algorithms including the recent `GRAC` [7] and 4 NMF-based approaches (`GNMF-A`, `GNMF-C`, `GNMF-L` [2, 7] and `JNMF` [6]);
- `ACMin-C` and `ACMin-S`, obtained by applying an attributed graph clustering method `ACMin` [42] over the graphs reduced from hypergraphs by clique expansion and star expansion, respectively;
- the `k-means` algorithm applied to the node attribute matrix.

For all competitors, we adopt the default parameter settings as suggested in their respective papers. As for `AHCKA`, unless otherwise specified, we set parameters on all datasets: $\alpha = 0.2$, $\beta = 0.5$, and $\gamma = 3$, parameter $K = 10$ for KNN construction, the convergence threshold $\epsilon_Q = 0.005$, and the numbers of iterations $T_a = 1000$, $T_i = 25$. The interval parameter τ is set to 5 on all datasets except the large and dense hypergraph Amazon, where we set $\tau = 1$ to expedite early termination in light of the immense per-iteration overhead when processing Amazon. On large datasets (i.e., Amazon and MAG-PM), T_i is set to 1 and $\beta = 0.4$.

Table 2. Clustering Quality on Small Datasets

Algorithm	Query				Cora-CA				Cora-CC				Citeseer			
	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI
HyperAdj	0.212	0.198	0.013	-0.004	0.233	0.216	0.038	0.022	0.255	0.191	0.039	0.015	0.226	0.182	0.008	0.002
HNCut	0.239	0.218	0.016	0.002	0.238	0.127	0.023	-0.002	0.213	0.125	0.021	-0.005	0.222	0.167	0.010	0.004
KaHyPar	0.220	0.205	0.016	0.003	0.275	0.265	0.084	0.050	0.309	0.289	0.135	0.089	0.275	0.265	0.045	0.036
k-means	0.586	0.581	0.461	0.230	0.349	0.297	0.158	0.086	0.351	0.312	0.176	0.097	0.460	0.424	0.219	0.185
ATHyperAdj	0.281	0.259	0.036	0.019	0.255	0.232	0.061	0.032	0.262	0.238	0.061	0.035	0.218	0.198	0.010	0.005
ATHNCut	0.241	0.220	0.017	0.003	0.438	0.297	0.263	0.183	0.556	0.456	0.317	0.288	0.563	0.483	0.325	0.286
ATMetis	0.520	0.507	0.349	0.264	0.575	0.550	0.403	0.346	0.552	**0.529**	0.379	0.310	0.612	0.590	0.357	0.348
ATKaHyPar	0.243	0.225	0.025	0.009	0.528	0.477	0.316	0.260	0.529	0.480	0.299	0.246	0.551	0.532	0.304	0.276
ACMin-C	0.233	0.219	0.017	0.003	0.526	0.493	0.319	0.237	0.556	0.473	0.349	0.259	0.643	0.587	0.355	0.376
ACMin-S	0.241	0.140	0.008	-0.002	0.523	0.477	0.318	0.239	0.526	0.462	0.340	0.249	0.636	0.597	0.351	0.365
GNMF-A	0.243	0.127	0.051	0.014	0.513	0.444	0.273	0.218	0.433	0.399	0.212	0.169	0.507	0.452	0.247	0.247
GNMF-C	0.245	0.136	0.046	0.016	0.512	0.446	0.275	0.216	0.439	0.406	0.226	0.183	0.508	0.453	0.248	0.251
GNMF-L	0.451	0.413	0.345	0.247	0.460	0.412	0.240	0.165	0.436	0.355	0.194	0.132	0.500	0.462	0.271	0.257
JNMF	0.216	0.211	0.014	-0.001	0.494	0.443	0.286	0.216	0.453	0.426	0.230	0.178	0.543	0.518	0.246	0.242
GRAC	0.410	0.389	0.196	0.087	0.601	0.593	0.376	0.308	0.556	0.507	0.349	0.262	0.612	0.575	0.329	0.332
AHCKA	**0.715**	**0.662**	**0.645**	**0.571**	**0.651**	**0.608**	**0.462**	**0.406**	**0.592**	0.520	**0.412**	**0.338**	**0.662**	**0.615**	**0.392**	**0.397**

Table 3. Clustering Quality on Medium/Large Datasets

Algorithm	20News				DBLP				Amazon				MAG-PM			
	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI
HyperAdj	0.338	0.274	0.010	0.010	0.234	0.158	0.019	0.007	0.292	0.105	0.043	0.070	0.138	0.078	0.051	0.028
HNCut	0.683	0.561	0.373	0.387	0.279	0.113	0.020	0.009	0.310	0.032	0.001	0.000	0.253	0.022	0.005	0.001
KaHyPar	0.479	0.468	0.169	0.172	0.559	0.534	0.390	0.338	0.494	0.442	**0.694**	0.385	0.367	0.306	0.483	0.247
k-means	0.404	0.373	0.147	0.045	0.529	0.513	0.362	0.283	0.380	0.257	0.362	0.175	0.272	0.196	0.229	0.071
ATHyperAdj	0.317	0.261	0.016	0.006	0.296	0.220	0.068	0.035	0.273	0.103	0.050	0.057	0.189	0.048	0.043	-0.006
ATHNCut	0.338	0.133	0.002	0.001	0.458	0.245	0.386	0.173	0.310	0.033	0.003	0.000	0.269	0.035	0.035	-0.001
ATMetis	0.612	0.596	0.264	0.281	0.671	0.670	0.567	0.496	OOM				0.304	0.254	0.401	0.196
ATKaHyPar	0.632	0.610	0.295	0.328	0.650	0.658	0.522	0.457	0.527	0.504	0.680	0.386	0.352	0.295	0.411	0.205
ACMin-C	0.558	0.524	0.219	0.239	0.607	0.563	0.503	0.445	0.458	0.113	0.354	0.244	0.519	0.293	0.499	0.430
ACMin-S	0.7116	**0.669**	0.365	0.416	0.547	0.474	0.472	0.359	0.473	0.056	0.393	0.263	0.550	0.341	0.550	**0.499**
GNMF-A	0.336	0.126	0.000	0.000	0.688	0.649	0.531	0.486	OOM				OOM			
GNMF-C	0.336	0.126	0.000	0.000	0.637	0.584	0.479	0.440	OOM				OOM			
GNMF-L	0.436	0.271	0.070	0.061	0.613	0.506	0.417	0.407	OOM				OOM			
JNMF	0.582	0.423	0.247	0.241	0.618	0.588	0.447	0.396	OOM				OOM			
GRAC	0.391	0.306	0.068	0.056	0.648	0.657	0.563	0.487	0.612	0.488	0.625	0.486	0.398	0.315	0.386	0.197
AHCKA	**0.7118**	0.658	**0.409**	**0.469**	**0.797**	**0.774**	**0.632**	**0.632**	**0.660**	**0.492**	0.630	**0.524**	**0.566**	**0.405**	**0.561**	0.471

7.2 Performance Evaluation

In this section, we present the experimental results on clustering quality and efficiency of all methods on all datasets. For each method, we repeat 10 times and report the average performance.

7.2.1 **Quality Evaluation.** The clustering quality is measured by four classic metrics including overall accuracy (Acc), average per-class F1 score (F1), normalized mutual information (NMI), and adjusted Rand index (ARI). Note that the former three metrics are in the range $[0, 1]$, whereas ARI ranges from -0.5 to 1. Tables 2 and 3 present the Acc, F1, NMI, and ARI scores of each method on small and medium/large datasets, respectively. The first observation from Tables 2 and 3 is

Table 4. Running Time of AHC Algorithms (Time in Seconds, RAM in GBs)

Algorithm	Query		Cora-CA		Cora-CC		Citeseer		20News		DBLP		Amazon		MAG-PM	
	Time	RAM	Time	RAM	Time	RAM	Time	RAM	Time	RAM	Time	RAM	Time	RAM	Time	RAM
GNMF-A	2.423	0.422	16.47	0.334	21.17	0.327	74.02	0.577	29.13	4.173	729.7	33.18	OOM		OOM	
GNMF-C	2.150	0.420	21.04	0.333	21.08	0.327	73.48	0.547	26.80	4.149	711.8	27.20	OOM		OOM	
GNMF-L	2.851	0.494	15.92	0.369	20.55	0.316	72.96	0.562	36.76	4.273	612.3	33.27	OOM		OOM	
JNMF	3.366	0.494	7.754	0.369	22.66	0.317	61.32	0.549	253.3	4.273	3247	33.27	OOM		OOM	
GRAC	1.701	**0.142**	7.661	0.288	3.696	0.287	13.15	0.454	**3.368**	**0.275**	91.21	1.700	14662	175.1	3504	92.69
AHCKA	**0.342**	0.161	**0.402**	**0.231**	**0.416**	**0.232**	**0.635**	**0.317**	8.176	0.383	**41.50**	**0.998**	**1286**	**56.71**	**1371**	**59.25**

that AHCKA consistently achieves superior performance over all competitors on all datasets under almost all metrics, often by a significant margin. Specifically, on all the four small datasets (*i.e.*, Query, Cora-CA, Cora-CC, and Citeseer), AHCKA outperforms the best competitors (underlined in Table 2) by at least 1.9%, 3.7% and 2.1% in terms of Acc, NMI, and ARI, respectively. On all the four medium/large attributed hypergraphs (*i.e.*, 20News, DBLP, Amazon, and MAG-PM), AHCKA also yields remarkable improvements upon the competitors, with percentages up to 10.9%, 10.4%, 6.5%, 13.8% in Acc, F1, NMI, and ARI respectively. Few exceptions exist, where AHCKA still leads in three out of the four metrics, demonstrating the best overall performance. The results in Tables 2 and 3 also confirm the effectiveness of AHCKA over various attributed hypergraphs from different application domains, *e.g.*, web queries, news messages, and review data. The superiority of AHCKA is ascribed to our carefully-designed optimizations based on KNN augmentation and MHC in Section 3 and Section 4, and the framework for generating high-quality BCM matrices in Section 5. In contrast, the baseline methods either overlook attribute information or fail to incorporate the topology and attribute information of input hypergraphs via multi-hops, and thus, produce inferior result quality.

7.2.2 **Efficiency Evaluation.** For a fair comparison, this set of experiments evaluates the empirical efficiency and memory overheads (RAM) against 5 AHC solutions, i.e., GNMF-A, GNMF-C, GNMF-L, JNMF, and GRAC. Table 4 reports the running time (measured in seconds, with KNN construction included) and memory overheads (measured in Gigabytes) required by each method on all datasets. We omit any methods with out-of-memory (OOM) errors. We can observe that AHCKA is significantly faster than all AHC methods on most datasets, often by orders of magnitude. For example, on a small graph Citeseer, AHCKA takes 0.635 seconds, while the fastest AHC competitor GRAC needs 13.15 seconds, meaning that AHCKA is 20.7× faster. On large attributed hypergraphs including Amazon and MAG-PM, most existing AHC solutions fail to finish due to the OOM errors, whereas AHCKA achieves 11.4× and 2.6× speedup over the only viable AHC competitor GRAC on Amazon and MAG-PM, respectively. The exception is on 20News that contains a paucity of hyperedges (100 hyperedges), where AHCKA is slower than GRAC. Recall that in Table 3, compared to AHCKA, GRAC yields far inferior accuracies in terms of clustering on 20News, which promotes the superiority of AHCKA over GRAC. As for the memory consumption (including the space to store hypergraphs), observe that AHCKA has comparable memory overheads with competitors on small graphs and up to 3.1× memory reduction on medium/large graphs.

7.3 Ablation Analysis

In this section, we experimentally study the effects of varying parameters and evaluate the effectiveness of our BCM initialization method InitBCM and the BCM conversion method Discretize.

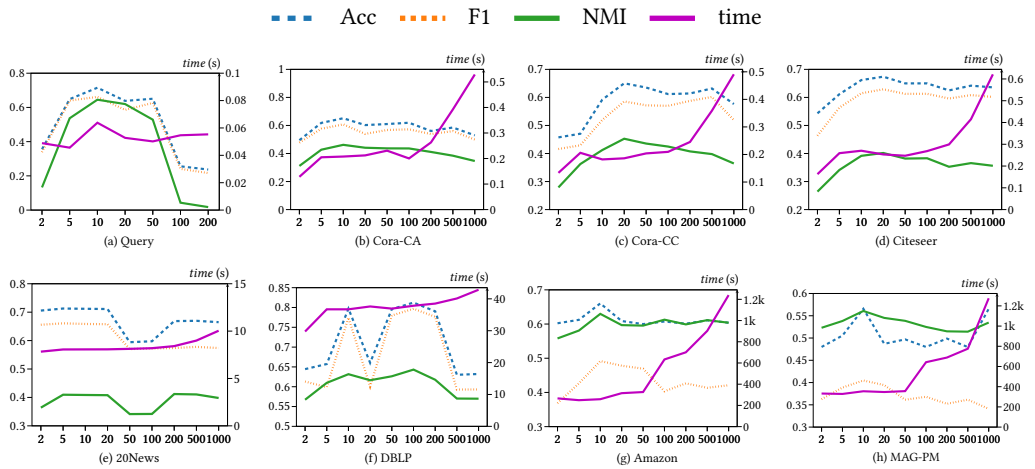


Fig. 4. Varying K (best viewed in color).

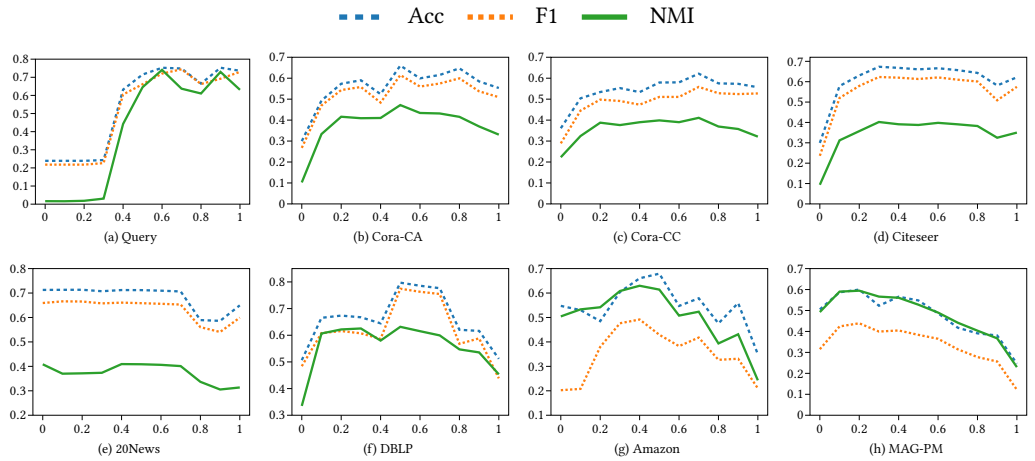


Fig. 5. Varying β (best viewed in color).

7.3.1 **Varying K.** Figure 4 depicts the Acc, F1, NMI scores, and the KNN computation time of AHCKA on 8 datasets when varying K from 2 to 1000. We can make the following observations. First, on most hypergraphs, the clustering accuracies of AHCKA first grow when K is increased from 2 to 10 and then decline, especially when K is beyond 50. The reasons are as follows. When K is small, the KNN graph $\mathcal{G}_K$ in AHCKA fails to capture the key information in the attribute matrix $\mathbf{X}$, leading to limited result quality. On the other hand, when K is large, more noisy or distorted information will be introduced in $\mathcal{G}_K$, and hence, causes accuracy loss. This coincides with our observation in the preliminary study in Figure 2. Moreover, as K goes up, the time of KNN construction increases considerably on all datasets. In sum, setting parameter K to 10 in AHCKA is a good choice on most datasets.

7.3.2 **Varying β.** Recall that in our (α, β, γ)-random walk model in Section 3.2, the parameter β is used to balance the combination of topological proximities from hypergraph $\mathcal{H}_O$ and the

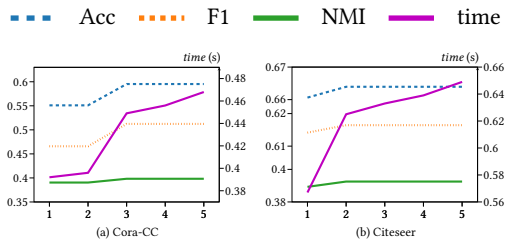


Fig. 6. Varying γ (best viewed in color).

Table 5. Ablation Analysis (Time in Seconds)

Algorithm	Query				Cora-CA				Cora-CC				Citeseer			
	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time
AHCKA-random-init	0.678	**0.662**	0.599	0.393	0.611	0.529	0.438	0.445	0.539	0.493	0.377	0.495	0.567	0.485	0.320	0.694
AHCKA-k-means	0.358	0.353	0.148	0.994	0.572	0.478	0.418	0.782	0.571	0.461	0.400	0.933	0.570	0.469	0.338	1.164
AHCKA	**0.715**	**0.662**	**0.645**	**0.342**	**0.651**	**0.608**	**0.462**	**0.402**	**0.592**	**0.520**	**0.412**	**0.416**	**0.662**	**0.615**	**0.392**	**0.635**
Algorithm	**20News**				**DBLP**				**Amazon**				**MAG-PM**			
	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time
AHCKA-random-init	0.625	0.609	0.361	10.543	0.637	0.603	0.585	**41.00**	0.623	0.297	0.562	1310	0.512	0.396	0.518	**881.7**
AHCKA-k-means	0.398	0.360	0.101	9.828	0.652	0.617	0.605	43.11	0.567	0.227	0.558	1492	0.536	0.276	0.504	4437
AHCKA	**0.7118**	**0.658**	**0.409**	**8.176**	**0.797**	**0.774**	**0.632**	41.50	**0.660**	**0.492**	**0.630**	**1286**	**0.566**	**0.405**	**0.561**	1371

attribute similarities from KNN graph $\mathcal{G}_K$. Figure 5 displays the clustering performance of AHCKA on 8 datasets when β is varied from 0 to 1. When $\beta = 0$, AHCKA degrades to a hypergraph clustering method without the consideration of any attribute information, whereas AHCKA only cluster the KNN graph $\mathcal{G}_K$ regardless of the topology structure in $\mathcal{H}$ if $\beta = 1$. From Figure 5, we can see a large β (e.g., 0.7-0.8) on small/medium datasets (Query, Cora-CA, Cora-CC, Citeseer, 20News, and DBLP) bring more performance enhancements, meaning that attribute information plays relatively more important roles on those datasets. This is because they have limited amounts of connections (or are too dense to be informative, e.g., on Query) in the original hypergraph structure as listed in Table 1 and rely on attribute similarities from the augmented KNN graph $\mathcal{G}_K$ for improved clustering. By contrast, on Amazon and MAG-PM, AHCKA achieves best clustering quality scores with small β in $[0.1, 0.4]$, which indicates that graph topology has higher weights on Amazon and MAG-PM.

7.3.3 **Varying γ.** We evaluate AHCKA in terms of clustering quality and running time when varying γ. Figure 6 displays the Acc, F1, NMI, and time on two representative datasets when γ is varied from 1 to 5. The results on other datasets are qualitatively similar and thus are omitted for the interest of space. It can be observed that in practice the Acc, F1, and NMI scores obtained by AHCKA first increase and then remain stable when γ is beyond 3 and 2 on Cora-CC and Citeseer, respectively. By contrast, the running time goes up as γ increases. Therefore, we set $\gamma = 3$ in experiments.

7.3.4 **Effectiveness Evaluation of `InitBCM` and `Discretize`.** To verify the effectiveness of our method `InitBCM` for the BCM initialization, we compare AHCKA with the ablated version AHCKA-random-init, where the BCM matrix $\mathbf{Y}^{(0)}$ is initialized at random. As shown in Table 5, AHCKA obtains remarkable improvements over AHCKA-random-init in Acc, F1, and NMI scores using comparable processing time. For instance, on Amazon, AHCKA outperforms AHCKA-random-init by a large margin of 3.7% Acc, 19.5% F1, and 6.8% NMI with 24 seconds less to process. On MAG-PM dataset, AHCKA needs additional running time compared to AHCKA-random-init. The reason is that AHCKA-random-init starts with a low-quality BCM and converges to local optimum solutions with suboptimal

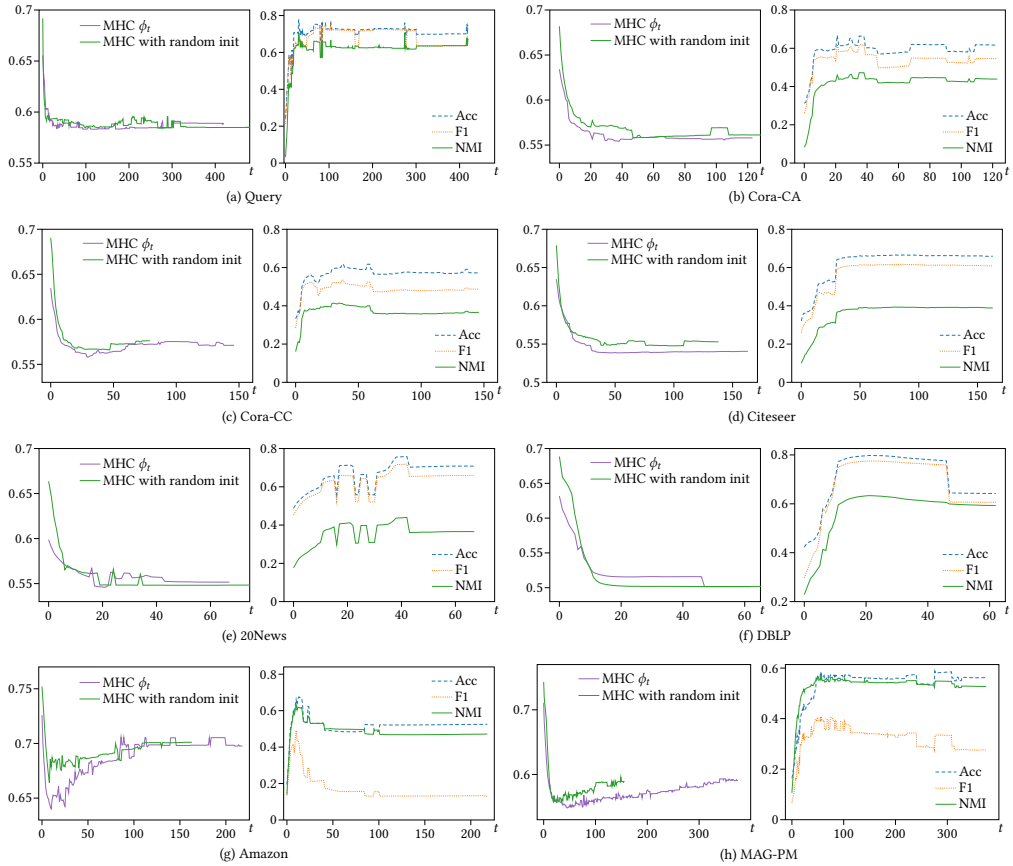


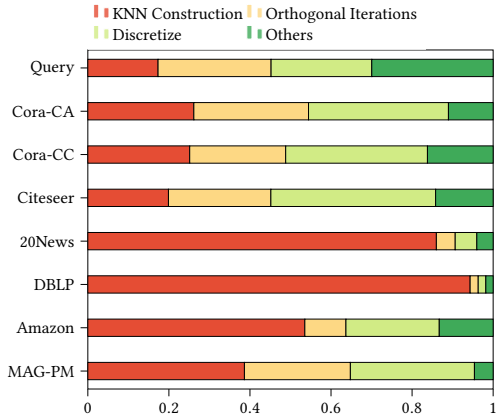
Fig. 7. Convergence Analysis (best viewed in color).

MHC, whereas AHCKA can bypass such pitfalls with a good initial BCM from InitBCM and continue searching for the optimal solution with more iterations, which in turn results in a considerable gap in clustering quality. In addition, we validate the effectiveness of Discretize used in AHCKA to transform k leading eigenvectors $\mathbf{Q}$ to BCM matrix $\mathbf{Y}$. Table 5 also reports the clustering accuracies of our complete AHCKA and a variant AHCKA-k-means obtained by replacing Discretize in AHCKA with k-means on all datasets. It can be observed that compared with AHCKA-k-means, AHCKA is able to output high-quality BCM matrices $\mathbf{Y}$ with substantially higher clustering accuracy scores while being up to 3.2× faster.

7.4 Convergence Analysis

We provide an empirical analysis pertinent to the convergence of AHCKA. To do so, we first disable the early termination strategies at Line 10 in Algorithm 1. We also set $\tau = 1$ so as to evaluate the MHC (denoted as ψ_t) of the BCM matrix $\mathbf{Y}^{(t)}$ generated in each t-th iteration of AHCKA and AHCKA-random-init, where t starts from 0 till convergence. Furthermore, we calculate the Acc, F1, and NMI scores with the ground truth for each BCM matrix $\mathbf{Y}^{(t)}$ generated throughout the iterative procedures of AHCKA. Figure 7 shows the MHC ϕ_t, Acc, F1, and NMI scores based on the BCM matrix of each iteration in AHCKA, as well as the MHC of AHCKA-random-init over all datasets. Notably, MHC ϕ_t experiences a sharp decline when t increases from 0 to 50 on most hypergraphs,

Fig. 8. Runtime Analysis (best viewed in color)



while the Acc, F1, and NMI results have significant growth. Moreover, compared to MHC with random init, MHC curves of AHCKA are mostly lower (better) on all datasets under the same t-th iteration. These phenomena demonstrate the effectiveness of our greedy initialization method InitBCM in facilitating fast convergence of AHCKA. However, when we keep increasing t, these scores either remain stable or deteriorate. For instance, MHC scores grow significantly after 10 iterations on Amazon, while there is a big drop in Acc and F1 scores when $t \geq 45$ on DBLP. This observation indicates that adding more iterations does not necessarily ensure better solutions but might impair results. Hence, the early termination strategies proposed in AHCKA can serve as an effective approach to remedy this issue.

7.5 Runtime Analysis

Figure 8 reports the percentages of time costs by the KNN construction in the preprocessing, orthogonal iterations, Discretize, and others in AHCKA. On small hypergraphs, the online stage consisting of orthogonal iterations, Discretize, and other operations consumes the majority of the computation time, whereas, on medium or large datasets, the preprocessing cost for KNN construction is much higher, especially on 20News and DBLP. Apart from KNN construction, another main cost arises from Discretize. In comparison, we observe that the orthogonal iterations in AHCKA have a relatively low cost. This is attributed to our non-trivial speedup optimizations including the decoupled sparse matrix multiplications in Section 5.1, greedy BCM initialization in Section 5.2, and early termination tricks validated in Section 7.4.

8 CONCLUSION

This paper presents AHCKA, an effective method for AHC computation that scales to large hypergraphs with millions of nodes while achieving state-of-the-art clustering quality. The superiority of AHCKA over existing solutions in terms of both efficiency and effectiveness is attributed to: (i) a KNN augmentation strategy on hypergraphs to judiciously exploit useful attribute information, (ii) a novel problem formulation based on a random walk model, and (iii) a highly efficient iterative optimization framework with speedup techniques. In the future, we plan to further develop a distributed version of AHCKA and extend AHCKA to cope with evolving attributed hypergraphs where node-attribute associations and connections between nodes change over time.

9 APPENDIX

9.1 Proofs

Proof of Lemma 4.1. Let $\widehat{\mathbf{W}}$ denote $h(\mathbf{W})$, i.e., $(\mathbf{Y}^\top\mathbf{Y})^{-1/2}\mathbf{Y}$ (Eq. (7)) , and we have

$$\Psi(\mathbf{W}) = \frac{1}{k} trace(\widehat{\mathbf{W}}^\top \mathbf{S}\widehat{\mathbf{W}}) = \frac{1}{k} trace(\mathbf{S}(\widehat{\mathbf{W}}\widehat{\mathbf{W}}^\top)) \tag{20}$$

Since $\widehat{\mathbf{W}}^\top\widehat{\mathbf{W}} = \mathbf{I}$, $\widehat{\mathbf{W}}$ is an $n \times k$ orthogonal matrix with rank equal to k. Based on basic matrix rank properties, the following inequalities regarding the rank of $\widehat{\mathbf{W}}\widehat{\mathbf{W}}^\top$ can be derived.

$$k = rank(\widehat{\mathbf{W}}) + rank(\widehat{\mathbf{W}}^\top) - k \leq rank(\widehat{\mathbf{W}}\widehat{\mathbf{W}}^\top) \leq \min(rank(\widehat{\mathbf{W}}), rank(\widehat{\mathbf{W}}^\top)) = k$$

It follows that $\widehat{\mathbf{W}}\widehat{\mathbf{W}}^\top$ is a symmetric matrix with rank k and the eigenvalue 0 of $\widehat{\mathbf{W}}\widehat{\mathbf{W}}^\top$ has multiplicity $n-k$. From the associativity of matrix multiplication, we have $(\widehat{\mathbf{W}}\widehat{\mathbf{W}}^\top)\widehat{\mathbf{W}} = \widehat{\mathbf{W}}(\widehat{\mathbf{W}}^\top\widehat{\mathbf{W}}) = \widehat{\mathbf{W}}$. Thus, these column vectors of $\widehat{\mathbf{W}}$ correspond to k unit eigenvectors of $\widehat{\mathbf{W}}\widehat{\mathbf{W}}^\top$ associated with eigenvalue 1. Since $\widehat{\mathbf{W}}\widehat{\mathbf{W}}^\top$ is a symmetric matrix, its k largest eigenvalues, and singular values are 1 while the other $n-k$ eigenvalues and singular values are 0. Because the row-stochastic matrix $\mathbf{S}$ is not necessarily Hermitian, we use Von Neumann's trace inequality to derive an upper bound of $\Psi(\mathbf{W})$ associated with the singular values of $\mathbf{S}$ and $\widehat{\mathbf{W}}\widehat{\mathbf{W}}^\top$.

$$\Psi(\mathbf{W}) \leq \frac{1}{k}(\sum_{i=1}^{k} 1 \cdot \sigma_i + \sum_{i=k+1}^{n} 0 \cdot \sigma_i) = \frac{1}{k}\sum_{i=1}^{k} 1 \cdot \sigma_i = \psi_\sigma \tag{21}$$

Aside from the k largest singular values, all the other singular values of $\widehat{\mathbf{W}}\widehat{\mathbf{W}}^\top$ are 0. Therefore, the resulting upper bound ψ_σ is the arithmetic mean of k largest singular values of $\mathbf{S}$. □

Proof of Lemma 4.2. Denote the eigenvector associated with the i-th largest eigenvalue λ'_i of $\mathbf{P}$ by e_i. Then $\mathbf{Q}$ is the matrix containing column vectors $e_2, \ldots, e_{k+1}$. For e_i where $1 \leq j \leq k$:

$$\mathbf{S}e_i = (\alpha \textstyle\sum_{l=0}^{\gamma}(1-\alpha)^l\mathbf{P}^l)e_i = (\alpha \sum_{l=0}^{\gamma}(1-\alpha)^l\lambda_i'^l)e_i \tag{22}$$

Hence, e_i is also an eigenvector of matrix $\mathbf{S}$ associated with eigenvalue $f(\lambda'_i) = \alpha \sum_{l=0}^{\gamma}(1-\alpha)^l\lambda_i'^l$. Because function $f(\lambda'_j)$ monotonously increases for $0 \leq \lambda'_i \leq 1$, we have $f(\lambda'_i) \leq f(\lambda'_j)$ if $\lambda'_i \leq \lambda'_j$ (under the assumption that $\lambda'_i \leq 0$). Therefore, $e_2, \ldots, e_{k+1}$ are also the second to $(k+1)$-th leading eigenvectors of $\mathbf{S}$ and we obtain $\Psi(\mathbf{Q}) = \frac{1}{k} trace(\mathbf{Q}^\top\mathbf{S}\mathbf{Q}) = \frac{1}{k}\sum_{i=2}^{k+1} e_i^\top\mathbf{S}e_i = \frac{1}{k}\sum_{i=2}^{k+1}\lambda_i = \psi_\lambda$, completing the proof. □

9.2 Connection between MHC and hypergraph normalized Cut

It has been shown [47] that hypergraph normalized cut (HNCut) is equivalent to the random walk conductance $\Phi_{\mathcal{H}}$ in Eq. (23):

$$\Phi_{\mathcal{H}}(\{C_1, \ldots, C_k\}) = \sum_{C \in \{C_1,\ldots,C_k\}} \sum_{v_i \in C_i} p_s(v_i) \sum_{v_j \notin C_i} \mathbf{P}_{\mathcal{H}}[i,j], \tag{23}$$

where $\mathbf{P}_{\mathcal{H}} = \mathbf{D}_V^{-1}\mathbf{A}^\top\mathbf{D}_E^{-1}\mathbf{A}$ denotes the transition probability matrix of random walk over hypergraph. $\Phi_{\mathcal{H}}$ equals to the average probability of transitioning to another cluster from a node drawn from the stationary distribution p_s of random walk. Note that in the above formula, there is a stationary distribution p_s, which is different from ours. Moreover, $\Phi_{\mathcal{H}}$ solely considers hypergraph structure for clustering. On the other hand, we propose a multi-hop conductance for attributed hypergraph clustering, so that node attribute similarity relations and higher-level structure are taken into account. In experiments, our solution outperforms `HNCut` by a significant margin, demonstrating that only utilizing hypergraph structure leads to inferior clustering performance.

ACKNOWLEDGMENTS

This work is supported by Hong Kong RGC ECS No. 25201221, National Natural Science Foundation of China No. 62202404, and project P0036831. Renchi Yang is supported by A*STAR, Singapore under Grant A19E3b0099 and the Departmental Start-up Fund from the Department of Computer Science, HKBU. This work is also supported by a research grant from Tencent Technology (Shenzhen) Co., Ltd (P0039546).

REFERENCES

[1] Zeyuan Allen Zhu, Silvio Lattanzi, and Vahab Mirrokni. 2013. A Local Algorithm for Finding Well-Connected Clusters. In *ICML*, Vol. 28. 396–404.
[2] Deng Cai, Xiaofei He, Jiawei Han, and Thomas S Huang. 2010. Graph regularized nonnegative matrix factorization for data representation. *IEEE transactions on pattern analysis and machine intelligence* 33, 8 (2010), 1548–1560.
[3] Yaoming Cai, Zijia Zhang, Zhihua Cai, Xiaobo Liu, and Xinwei Jiang. 2022. Hypergraph-Structured Autoencoder for Unsupervised and Semisupervised Classification of Hyperspectral Image. *IEEE Geosci. Remote. Sens. Lett.* 19 (2022), 1–5.
[4] T.-H. Hubert Chan and Zhibin Liang. 2018. Generalizing the Hypergraph Laplacian via a Diffusion Process with Mediators. *arXiv:1804.11128 [cs]* (2018).
[5] Hong Cheng, Yang Zhou, and Jeffrey Xu Yu. 2011. Clustering large attributed graphs: A balance between structural and attribute similarities. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 5, 2 (2011), 1–33.
[6] Rundong Du, Barry Drake, and Haesun Park. 2019. Hybrid Clustering Based on Content and Connection Structure Using Joint Nonnegative Matrix Factorization. *Journal of Global Optimization* 74, 4 (2019), 861–877.
[7] Barakeel Fanseu Kamhoua, Lin Zhang, Kaili Ma, James Cheng, Bo Li, and Bo Han. 2021. HyperGraph Convolution Based Attributed HyperGraph Clustering. In *CIKM*. 453–463.
[8] Thomas Gaudelet, Noël Malod-Dognin, and Natasa Przulj. 2018. Higher-order molecular organization as a source of biological function. *Bioinform.* 34, 17 (2018), i944–i953.
[9] Lars Gottesbüren, Tobias Heuer, and Peter Sanders. 2022. Parallel Flow-Based Hypergraph Partitioning. In *SEA*, Vol. 233. 5:1–5:21.
[10] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *ICML*, Vol. 119. 3887–3896.
[11] Koby Hayashi, Sinan G. Aksoy, Cheong Hee Park, and Haesun Park. 2020. Hypergraph Random Walks, Laplacians, and Clustering. In *CIKM*. 495–504.
[12] Matthias Hein, Simon Setzer, Leonardo Jost, and Syama Sundar Rangapuram. 2013. The Total Variation on Hypergraphs - Learning on Hypergraphs Revisited. In *NeurIPS*, Vol. 26.
[13] Ling Huang, Chang-Dong Wang, and Philip S. Yu. 2021. Higher Order Connection Enhanced Community Detection in Adversarial Multiview Networks. *IEEE Transactions on Cybernetics* (2021), 1–15.
[14] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*. 271–279.
[15] Jinhong Jung, Namyong Park, Sael Lee, and U Kang. 2017. BePI: Fast and Memory-Efficient Method for Billion-Scale Random Walk with Restart. In *SIGMOD*. 789–804.
[16] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. 1999. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 7, 1 (1999), 69–79.
[17] George Karypis and Vipin Kumar. 1998. Multilevelk-Way Partitioning Scheme for Irregular Graphs. *J. Parallel and Distrib. Comput.* 48, 1 (1998), 96–129.
[18] Sungwoong Kim, Sebastian Nowozin, Pushmeet Kohli, and Chang Yoo. 2011. Higher-Order Correlation Clustering for Image Segmentation. In *NeurIPS*, Vol. 24.
[19] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[20] Tarun Kumar, Sankaran Vaidyanathan, Harini Ananthapadmanabhan, Srinivasan Parthasarathy, and Balaraman Ravindran. 2018. Hypergraph Clustering: A Modularity Maximization Approach. *arXiv:1812.10869 [cs, stat]* (2018).
[21] Pan Li and Olgica Milenkovic. 2018. Submodular Hypergraphs: p-Laplacians, Cheeger Inequalities and Spectral Clustering. In *ICML*. 3014–3023.
[22] S. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–137.
[23] Christopher Lueg. 2003. *From Usenet to CoWebs: interacting with social information spaces*. Springer Science & Business Media.
[24] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *EMNLP-IJCNLP*. 188–197.

[25] Haekyu Park, Jinhong Jung, and U. Kang. 2017. A comparative study of matrix factorization and random walk with restart in recommender systems. In *2017 IEEE International Conference on Big Data (Big Data)*. 756–765.
[26] Matthew J. Rattigan, Marc Maier, and David Jensen. 2007. Graph Clustering with Network Structure Indices. In *ICML*. 783–790.
[27] J.A. Rodri´guez. 2002. On the Laplacian Eigenvalues and Metric Parameters of Hypergraphs. *Linear and Multilinear Algebra* 50, 1 (2002), 1–14.
[28] Y SAAD. 1992. Numerical Methods for Large Eigenvalue Problems. *Algorithms and Architectures for Advanced Scientific Computing* (1992).
[29] Sebastian Schlag, Tobias Heuer, Lars Gottesbüren, Yaroslav Akhremtsev, Christian Schulz, and Peter Sanders. 2022. High-Quality Hypergraph Partitioning. *ACM J. Exp. Algorithmics* (2022).
[30] Jianbo Shi and Jitendra Malik. 2000. Normalized Cuts and Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 22, 8 (2000), 888–905.
[31] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Paul Hsu, and Kuansan Wang. 2015. An Overview of Microsoft Academic Service (MAS) and Applications. In *WWW*. 243–246.
[32] Hwanjun Song, Jae-Gil Lee, and Wook-Shin Han. 2017. PAMAE: Parallel k-Medoids Clustering with High Accuracy and Efficiency. In *KDD*. 1087–1096.
[33] Yuuki Takai, Atsushi Miyauchi, Masahiro Ikeda, and Yuichi Yoshida. 2020. Hypergraph Clustering Based on PageRank. In *KDD*. 1970–1978.
[34] Hanghang Tong, Christos Faloutsos, and Jia-yu Pan. 2006. Fast Random Walk with Restart and Its Applications. In *ICDM*. 613–622.
[35] Nate Veldt, Austin R. Benson, and Jon Kleinberg. 2022. Hypergraph Cuts with General Splitting Functions. *SIAM Rev.* 64, 3 (2022), 650–685.
[36] Joyce Jiyoung Whang, Rundong Du, Sangwon Jung, Geon Lee, Barry Drake, Qingqing Liu, Seonggoo Kang, and Haesun Park. 2020. MEGA: Multi-View Semi-Supervised Clustering of Hypergraphs. *Proceedings of the VLDB Endowment* 13, 5 (2020), 698–711.
[37] Joong-Ho Won, Hua Zhou, and Kenneth Lange. 2021. Orthogonal trace-sum maximization: Applications, local algorithms, and global optimality. *SIAM J. Matrix Anal. Appl.* 42, 2 (2021), 859–882.
[38] Lei Wu, Yufeng Hu, Yajin Zhou, Haoyu Wang, Xiapu Luo, Zhi Wang, Fan Zhang, and Kui Ren. 2021. Towards Understanding and Demystifying Bitcoin Mixing Services. In *WWW*. 33–44.
[39] Ming-Juan Wu, Ying-Lian Gao, Jin-Xing Liu, Chun-Hou Zheng, and Juan Wang. 2020. Integrative Hypergraph Regularization Principal Component Analysis for Sample Clustering and Co-Expression Genes Network Analysis on Multi-Omics Data. *IEEE Journal of Biomedical and Health Informatics* 24, 6 (2020), 1823–1834.
[40] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. 2012. A Model-Based Approach to Attributed Graph Clustering. In *SIGMOD*. 505–516.
[41] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. 2019. HyperGCN: a new method of training graph convolutional networks on hypergraphs. In *NeurIPS*. Number 135. 1511–1522.
[42] 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.
[43] Tianbao Yang, Rong Jin, Yun Chi, and Shenghuo Zhu. 2009. Combining Link and Content for Community Detection: A Discriminative Approach. In *KDD*. 927–936.
[44] Stella X. Yu and Jianbo Shi. 2003. Multiclass Spectral Clustering. In *ICCV*. 313.
[45] Raphael Yuster and Uri Zwick. 2005. Fast sparse matrix multiplication. *ACM Transactions On Algorithms (TALG)* 1, 1 (2005), 2–13.
[46] Xiaotong Zhang, Han Liu, Qimai Li, and Xiao-Ming Wu. 2019. Attributed Graph Clustering via Adaptive Graph Convolution. In *IJCAI*. 4327–4333.
[47] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. 2007. Learning with Hypergraphs: Clustering, Classification, and Embedding. In *NeurIPS*, Vol. 19.
[48] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2009. Graph clustering based on structural/attribute similarities. *Proceedings of the VLDB Endowment* 2, 1 (2009), 718–729.
[49] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2010. Clustering Large Attributed Graphs: An Efficient Incremental Approach. In *2010 IEEE International Conference on Data Mining*. 689–698.

Received October 2022; revised January 2023; accepted February 2023