The VLDB Journal (2024) 33:1913–1943
https://doi.org/10.1007/s00778-024-00875-8

REGULAR PAPER

# A versatile framework for attributed network clustering via K-nearest neighbor augmentation

Yiran Li[1] · Gongyao Guo[1] · Jieming Shi[1] · Renchi Yang[2] · Shiqi Shen[3] · Qing Li[1] · Jun Luo[4]

Received: 29 December 2023 / Revised: 8 June 2024 / Accepted: 7 August 2024 / Published online: 16 September 2024

**Abstract**
Attributed networks containing entity-specific information in node attributes are ubiquitous in modeling social networks, e-commerce, bioinformatics, etc. Their inherent network topology ranges from simple graphs to hypergraphs with high-order interactions and multiplex graphs with separate layers. An important graph mining task is node clustering, aiming to partition the nodes of an attributed network 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 for effective clustering on multiple types of attributed networks. In this paper, we first present AHCKA as an efficient approach to *attributed hypergraph clustering* (AHC). AHCKA includes a carefully-crafted $K$-nearest neighbor augmentation strategy for the optimized exploitation of attribute information on hypergraphs, a joint hypergraph random walk model to devise an effective AHC objective, and an efficient solver with speedup techniques for the objective optimization. The proposed techniques are extensible to various types of attributed networks, and thus, we develop ANCKA as a versatile attributed network clustering framework, capable of *attributed graph clustering*, *attributed multiplex graph clustering*, and AHC. Moreover, we devise ANCKA-GPU with algorithmic designs tailored for GPU acceleration to boost efficiency. We have conducted extensive experiments to compare our methods with 19 competitors on 8 attributed hypergraphs, 16 competitors on 6 attributed graphs, and 16 competitors on 3 attributed multiplex graphs, all demonstrating the superb clustering quality and efficiency of our methods.

**Keywords** Clustering · Attributed Graph · Random Walks · KNN · GPU Computing

## 1 Introduction

An attributed network contains a network topology with attributes associated with nodes. Representative types of attributed networks include attributed graphs, attributed hypergraphs, and attributed multiplex graphs. Given an attributed network $\mathcal{N}$, node clustering is an important task in graph mining, which aims to divide the $n$ nodes of $\mathcal{N}$ into $k$ disjoint clusters, such that nodes within the same cluster are close to each other in the network topology and similar to each other in terms of attribute values. Clustering on attributed networks finds important applications in biological analysis [1], online marketing [2], social network [3, 4], Web analysis [5], etc.

✉ Jieming Shi
jieming.shi@polyu.edu.hk

Yiran Li
yi-ran.li@connect.polyu.hk

Gongyao Guo
gongyao.guo@connect.polyu.hk

Renchi Yang
renchi@hkbu.edu.hk

Shiqi Shen
shiqishen@tencent.com

Qing Li
csqli@comp.polyu.edu.hk

Jun Luo
jluo@lscm.hk

1 The Hong Kong Polytechnic University, Hung Hom, Hong Kong SAR, China

2 Hong Kong Baptist University, Kowloon Tong, Hong Kong SAR, China

3 WeChat Tencent, Beijing, China

4 Logistics and Supply Chain MultiTech R&D Centre, Pok Fu Lam, Hong Kong

In this work, we present ANCKA, an effective and efficient attributed network clustering method that is versatile to support attributed hypergraph clustering (AHC), attributed graph clustering (AGC), and attributed multiplex graph clustering (AMGC). ANCKA subsumes our previous work AHCKA [6] that is dedicated to AHC. In what follows, we first elaborate on AHC and then generalize to AGC and AMGC.

In a hypergraph, each edge can join an arbitrary number of nodes, referred to as a *hyperedge*. The hyperedge allows a precise description of multilateral relationships between nodes, such as collaboration relationships of multiple authors of a paper, interactions among proteins [7], products purchased together in one shopping cart, transactions involving multiple accounts [8]. In practice, nodes in hypergraphs are often associated with many attributes, e.g., the academic profile of authors and the descriptive data of products. The AHC problem 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 [9], organization structure detection [1], Web query analysis [5], biological analysis [10], etc. As another example, AHC can cluster together academic publications with high relevance by considering co-authorship hyperedges and keyword attributes in academic hypergraphs [11].

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 [12, 13], higher-order relationships between nodes and node-attribute associations are crucial for clustering. However, computing such multi-hop relationships and associations via hyperedges usually with more than two nodes in attributed hypergraphs is 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 [14–16] 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 [2, 13]. However, most of these works 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 [1, 5], 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 higher-order relationships between nodes, thereby limiting their result utility. The state-of-the-art approach GRAC [11] extends *graph convolution* [17] to hypergraphs, indirectly incorporating higher-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? We offer 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 surpasses existing solutions through several key techniques. The first one is a $K$-nearest 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 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 attributed hypergraph data in different domains. Compared with baselines, AHCKA exhibits superior performance in 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 state-of-the-art. Our work AHCKA has been published in [6].

In addition to attributed hypergraphs, attributed graphs and attributed multiplex graphs are prevalent in real-world scenarios, such as social networks [18] and citation networks [19]. Different from hypergraphs that allow more than two nodes to form an edge, in a graph, an edge connects exactly two nodes. A multiplex graph consists of multiple layers of graphs with a shared set of nodes, and different graph layers represent node connections from different perspectives or domains, e.g., different types of relationships or relations formed in different time frames or spaces [18, 19]. Attributed graph clustering (AGC) is one of the most significant graph mining problems, extensively studied in the literature [2, 13], with many applications, e.g., community detection in social networks [20] and functional cartography of metabolic networks [21]. Furthermore, a rich collection of studies on attributed multiplex graph clustering (AMGC) also exists in [22–25], to support important applications, e.g., biological analysis [19], community detection [18] and social analysis [26]. A previous general framework [27] relies on expensive graph convolutions to support various clustering tasks.

In this work, we extend AHCKA for AHC to a versatile framework ANCKA that can efficiently handle attributed Network clustering tasks (AHC, AGC, and AMGC) to produce high-quality clusters on large data. ANCKA inherits the powerful KNN augmentation scheme and the formulation of clustering objective in AHCKA. We further develop a generalized joint random walk model in ANCKA with proper transition matrices to support random walks on KNN augmented hypergraphs, graphs, and multiplex graphs simultaneously. Efficient optimization techniques are applied in ANCKA to retain the advantage of high efficiency for clustering. Despite the superior efficiency, clustering million-scale datasets with ANCKA can still take dozens of minutes. Moreover, after observing the limited speedup ratio by increasing the number of CPU threads used, we pinpoint the efficiency bottlenecks and design the GPU-accelerated ANCKA-GPU, to boost the efficiency to another level, especially on large-scale datasets. ANCKA-GPU consists of GPU-based optimization techniques and KNN construction procedures to speed up. We have conducted extensive experiments to compare ANCKA with 16 competitors on various attributed graphs and 16 competitors on attributed multiplex graphs. In all three tasks, ANCKA obtains superior performance regarding both clustering quality and efficiency. The GPU implementation ANCKA-GPU further reduces time costs significantly, often by an order of magnitude on large datasets.

We summarize the contributions of this work 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 with the objective of optimizing a quality measure based on a joint random walk model over the KNN augmented hypergraph.
- 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.
- We justify the application of KNN augmentation to various types of networks, generalize the techniques, and design a versatile method ANCKA to efficiently perform AHC, AGC, and AMGC and produce high-quality clusters.
- We develop ANCKA-GPU with customized GPU kernels to improve the efficiency further with a series of GPU-based optimizations while maintaining clustering quality.
- The excellent performance of ANCKA is validated by comprehensive experiments against 19 AHC competitors, 16 AGC competitors, and 16 AMGC competitors, over real-world datasets.

The remainder of this paper is structured as follows: Sect. 2 introduces the preliminaries of AHC, AGC, and AMGC. Section 3 outlines the KNN augmentation strategy and random walk scheme for AHC, along with the AHC clustering objective. Section 4 offers a theoretical analysis of the proposed AHC method AHCKA, while Sect. 5 details the algorithmic procedures of AHCKA. Section 6 presents the versatile ANCKA framework for AHC, AGC, and AMGC. Section 7 discusses GPU-based techniques for enhancing clustering efficiency in ANCKA-GPU. Section 8 provides a comprehensive experimental evaluation. Section 9 reviews relevant literature, and Sect. 10 concludes the paper.

## 2 Preliminaries

**Attributed Network.** Let $\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ be an attributed network, where $\mathcal{V}$ is the node set with cardinality $|\mathcal{V}| = n$, $\mathcal{E}$ is the edge (or hyperedge) set with cardinality $|\mathcal{E}| = m$, and $\mathbf{X} \in \mathbb{R}^{n\times d}$ represents a node attribute matrix. A node $v_j \in \mathcal{V}$ has degree $\delta(v_j)$, which is the number of edges (or 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 consider three types of attributed networks $\mathcal{N}$, including attributed hypergraphs $\mathcal{H}$, attributed graphs $\mathcal{G}$, and attributed multiplex graphs $\mathcal{G}_M$, characterized by different nature of $\mathcal{E}$.

**Attributed Hypergraph** is denoted by $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$. $\mathcal{E}$ is the set of $m$ hyperedges where each $e_i \in \mathcal{E}$ 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 $v_j \in e_i$. We denote by $\mathbf{H} \in \mathbb{R}^{m\times n}$ the incidence matrix of hypergraph $\mathcal{H}$, where each entry $\mathbf{H}[i, j] = 1$ if $v_j \in e_i$, otherwise $\mathbf{H}[i, j] = 0$. 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. Figure 1 shows an attributed hypergraph $\mathcal{H}$ with 8 nodes and 5 hyperedges, where each node has 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.

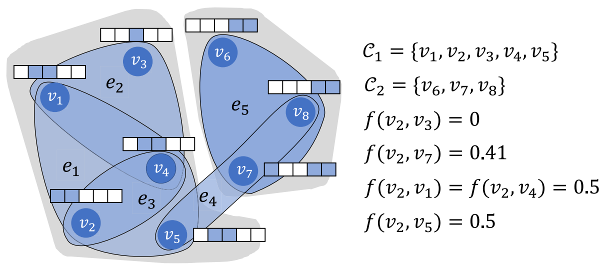

**Fig. 1** An Example Attributed Hypergraph

**Attributed Graph** is denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where every edge in $\mathcal{E}$ connects exactly two nodes. A graph $\mathcal{G}$ can be undirected or directed. An undirected edge can be viewed as two directed edges of the same node pair in reversed directions. Different from a hypergraph incident matrix between nodes and hyperedges, graph adjacency matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ encodes the structure of $\mathcal{G}$, where entry $\mathbf{A}[i, j]$ is 1 if there is an edge from node $v_i$ to node $v_j$, i.e., $(v_i, v_j) \in \mathcal{E}_G$, or 0 if otherwise. Let $\mathbf{D} \in \mathbb{R}^{n\times n}$ be the diagonal node degree matrix of $\mathcal{G}$.
**Attributed Multiplex Graph** is $\mathcal{G}_M = (\mathcal{V}, \mathcal{E}_1, ..., \mathcal{E}_L, \mathbf{X})$, consisting of $L$ graph layers. Every $l$-th layer has its own edge set $\mathcal{E}_l$, and can be viewed as an attributed graph $\mathcal{G}_l$ with $\mathcal{E}_l$, adjacency matrix $\mathbf{A}_l$, and diagonal node degree matrix $\mathbf{D}_l$.
**The Clustering Problem.** Given an attributed network $\mathcal{N}$ that can be $\mathcal{H}$, $\mathcal{G}$, or $\mathcal{G}_M$, we study the clustering problem that encompasses *attributed hypergraph clustering* (AHC), *attributed graph clustering* (AGC), and *attributed multiplex graph clustering* (AMGC). Given a specified number $k$ of clusters and an attributed network $\mathcal{N}$, the clustering task is to divide the node set $\mathcal{V}$ into $k$ disjoint subsets $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ such that $\bigcup_{i=1}^{k} \mathcal{C}_i = \mathcal{V}$ and the following properties are satisfied:

1. Nodes within the same cluster are closely connected to each other in the network structure, 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**).

For instance, when the input network $\mathcal{N}$ is the attributed hypergraph $\mathcal{H}$ in Fig. 1, $\mathcal{H}$ is partitioned into two clusters $\mathcal{C}_1$ and $\mathcal{C}_2$. We can observe that nodes $v_1$-$v_5$ in $\mathcal{C}_1$ share similar attributes and are closely connected to each other, whereas nodes $v_6$, $v_7$ and $v_8$ form a cluster $\mathcal{C}_2$ that is separated from $\mathcal{C}_1$ with a paucity of connections and distinct attributes.

## 3 Attributed hypergraph clustering

As mentioned, we first focus on attributed hypergraph clustering (AHC) and present our method AHCKA [6] in Sects. 3, 4, and 5. Specifically, we will devise a random walk scheme on a K-nearest neighbor augmented hypergraph and present the AHC objective in Sect. 3, conduct theoretical analysis to support the design of AHCKA in Sect. 4, and develop the algorithmic details of AHCKA in Sect. 5.

For the problem of AHC, a central challenge is how to simultaneously exploit both hypergraph structure and attribute information for improved clustering quality. In literature, it is a natural and effective approach to augment network structure with attribute similarity strengths [13, 28]. However, since a hypergraph yields different topological characteristics as illustrated in Fig. 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 (Sect. 8.3).

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 augmentation strategy is orthogonal to the topological nature of hypergraph, its application to other types of networks, such as attributed graphs and attributed multiplex graphs, will be explained shortly in Sect. 6. Then we formulate Attributed Hypergraph Clustering as *Augmented Hypergraph Clustering*, with the same abbreviation AHC. The augmented hypergraph involves both 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 design the $(\alpha, \beta, \gamma)$-*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 KNN augmentation

Although the vanilla augmentation strategy improves the clustering quality in attributed graphs [13, 28], 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-CA dataset[1] containing 2.7k academic papers in 7 research fields (i.e., 7 clusters). Every node has an attribute vector indicating the presence of words in the corresponding publication. 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 $K$-

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

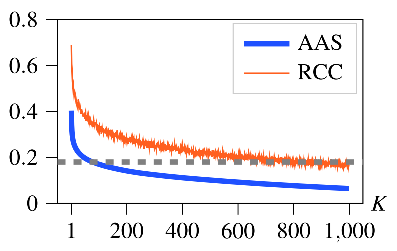

**Fig. 2** AAS and RCC on Cora-CA (best viewed in color)

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 Fig. 1, where nodes $v_2, v_3$ 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:

$$\mathbf{A}_K[i,j] = \begin{cases} 0, & \text{if } v_i \notin N_K(v_j) \text{ and } v_j \notin N_K(v_i), \\ 2 \cdot f(\mathbf{X}[i], \mathbf{X}[j]), & \text{if } v_i \in N_K(v_j) \text{ and } v_j \in N_K(v_i), \\ f(\mathbf{X}[i], \mathbf{X}[j]), & \text{otherwise.} \end{cases} \quad (1)$$

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 Fig. 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 Fig. 2). Therefore, such connections can be overlooked without impeding the clustering quality. Secondly, if we revisit the example in Fig. 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. 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 with $O(n \log n)$ complexity [29, 30].

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 Fig. 2. Meanwhile, $K$ cannot be too small (e.g., 5), or it will fail to utilize highly similar nodes that usually have high RCC. The trade-off of choosing $K$ is evaluated in Sect. 8.3.

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 $(\alpha, \beta, \gamma)$-random walk

Random walk with restart [31] (RWR) is one of the most common and effective models for capturing the multi-hop relationships between nodes in a graph [32], and is widely used in many tasks such as ranking [31, 33], recommendation [34], and clustering [35]. Given a graph $\mathcal{G}$, a source node $u$ and a stopping probability $\alpha$ (typically $\alpha = 0.2$), at each step, an RWR originating from $u$ either stops at the current node with probability $\alpha$, 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* [36]) 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, named $(\alpha, \beta, \gamma)$-random walk, which conducts the RWR process over $\mathcal{H}_O$ and $\mathcal{G}_K$ jointly to seamlessly integrate topological proximity over both networks. Definition 1 states the formal definition of the $(\alpha, \beta, \gamma)$-random walk process.

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

- With probability $\alpha$, $W$ 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 $\beta_i$, $W$ 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.

Each node $v_i$ is associated with a parameter $\beta_i$ (see Eq. (2)) used to control the joint navigation between hypergraph $\mathcal{H}_O$ and KNN $\mathcal{G}_K$. The larger $\beta_i$ is, the more likely that the random walk jumps to the neighbors of $v_i$ in KNN $\mathcal{G}_K$.

$$\beta_i = \begin{cases} 0, & \text{if } \mathbf{X}[i] \text{ is a zero vector;} \\ 1, & \text{else if } \delta(v_i) = 0; \\ \beta, & \text{otherwise.} \end{cases} \tag{2}$$

In general, we set $\beta_i$ to $\beta \in [0, 1]$, which is a user-specified parameter. In particular cases, 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 $\beta_i$ to 0. Conversely, $\beta_i$ 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 $(\alpha, \beta, \gamma)$-random walk from $v_i$ stopping at $v_j$ in the end. Based on Definition 1, we can derive the following formula for $s(v_i, v_j)$:

$$s(v_i, v_j) = \mathbf{S}[i, j] = \alpha \textstyle\sum_{\ell=0}^{\gamma} (1-\alpha)^{\ell} \mathbf{P}^{\ell}[i, j], \tag{3}$$

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

$$\mathbf{P} = (\mathbf{I} - \mathbf{B}) \cdot \mathbf{D}_V^{-1} \mathbf{H}^{\top} \mathbf{D}_E^{-1} \mathbf{H} + \mathbf{B} \mathbf{D}_K^{-1} \mathbf{A}_K, \tag{4}$$

$\mathbf{B} = diag(\beta_1, \ldots, \beta_n)$ is a diagonal matrix containing $\beta_i$ parameters, and $\mathbf{D}_K$ is the diagonal degree matrix of $\mathcal{G}_K$. $\mathbf{P}^{\ell}[i, j]$ is the probability that a $\ell$-hop 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 $\mathcal{C}$ 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 $(\alpha, \beta, \gamma)$-random walk $W$ from any node in $\mathcal{C}$, $W$ should have a low probability of escaping from $\mathcal{C}$, i.e., ending at any node outside $\mathcal{C}$. We refer to this escaping probability $\phi(\mathcal{C})$ as the *multi-hop conductance* (MHC) of $\mathcal{C}$, defined in Eq. (5).

$$\phi(\mathcal{C}) = \tfrac{1}{|\mathcal{C}|} \textstyle\sum_{v_i \in \mathcal{C}} \sum_{v_j \notin \mathcal{C}} s(v_i, v_j) \tag{5}$$

Since a low MHC $\phi(\mathcal{C})$ reflects a high coherence of cluster $\mathcal{C}$, we then formulate AHC as an optimization problem of finding $k$ clusters $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ such that their MHC $\Phi(\{\mathcal{C}_1, \ldots, \mathcal{C}_k\})$ (Eq. (6)) is minimized.

$$\Phi(\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}) = \frac{1}{k} \sum_{\mathcal{C} \in \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}} \frac{1}{|\mathcal{C}|} \sum_{v_i \in \mathcal{C}} \sum_{v_j \notin \mathcal{C}} s(v_i, v_j) \tag{6}$$

Directly minimizing Eq. (6) requires computing $s(v_i, v_j)$ (Eq. (3)) of every two nodes $v_i \in \mathcal{C}$, $v_j \in \mathcal{V} \setminus \mathcal{C}$, $\forall \mathcal{C} \in \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k\}$, which is prohibitively expensive due to intractable computation time (i.e., $O(n^3)$) and storage space (i.e., $O(n^2)$). In addition, the minimization of $\Phi(\{\mathcal{C}_1, \ldots, \mathcal{C}_k\})$ is an NP-complete combinatorial optimization problem [37], rendering the exact solution unattainable on large graphs.

## 4 Theoretical analysis for AHCKA

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. (6) 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 $\{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k\}$ on $\mathcal{H}$ satisfying $\bigcup_{i=1}^{k} \mathcal{C}_i = \mathcal{V}$, they can be represented by a binary matrix $\mathbf{Y} \in \{0, 1\}^{n \times k}$, where for each node $v_i$ and cluster $\mathcal{C}_j$

$$\mathbf{Y}[i, j] = \begin{cases} 1, & v_i \in \mathcal{C}_j \\ 0, & v_i \in \mathcal{V} \setminus \mathcal{C}_j. \end{cases} \tag{7}$$

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

$$h(\mathbf{Y}) = (\mathbf{Y}^{\top}\mathbf{Y})^{-1/2}\mathbf{Y} = \widehat{\mathbf{Y}} \tag{8}$$

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 $\{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k\}$ and their corresponding BCM matrix $\mathbf{Y}$, it is trivial to show

$$\Phi(\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}) = 1 - \Psi(\mathbf{Y}), \tag{9}$$

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

$$\Psi(\mathbf{Y}) = \frac{1}{k} trace(\widehat{\mathbf{Y}}^\top \mathbf{S} \widehat{\mathbf{Y}}). \tag{10}$$

Equation (9) suggests that the minimization of MHC $\Phi$ $(\{\mathcal{C}_1, \ldots, \mathcal{C}_k\})$ 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, instead of computing the exact solution, we utilize 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 $\psi_\sigma$ for $\Psi(\mathbf{Y})$.

**Lemma 1** *Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$ be the k largest singular values of matrix* $\mathbf{S}$ *in Eq. (3). 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 1 implies that if we can first find a fractional matrix $\mathbf{W}$ such that $\Psi(\mathbf{W})$ is close to $\psi_\sigma$, a high-quality BCM matrix $\mathbf{Y}$ can be converted from $\mathbf{W}$ by leveraging algorithms such as $k$-Means [38]. Although we can obtain such a fractional matrix $\mathbf{W}$ by applying trace maximization techniques [39] to Eq. (10), it still remains tenaciously challenging to compute $\mathbf{S}$. (All proofs are in the technical report [40].)

**Lemma 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. (4)). 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.*

We exclude the first eigenvector $\frac{1}{\sqrt{n}} \cdot \mathbf{1}$ of $\mathbf{P}$ as it is useless for clustering. By virtue of our analysis in Lemma 2, the second to $(k+1)$-th leading eigenvectors $\mathbf{Q}$ of $\mathbf{P}$ (see Eq. (4)) can be regarded as a rough $\mathbf{W}$ since $\Psi(\mathbf{Q}) = \psi_\lambda \leq \psi_\sigma$ and the gap between $\psi_\lambda$ and $\psi_\sigma$ is insignificant in practice. For instance, on the Cora-CA dataset, we can obtain $\psi_\sigma = 0.668$ and $\psi_\lambda = 0.596$ (*i.e.*, $\Phi_\sigma = 1 - \psi_\sigma = 0.332$, $\Phi_\lambda = 1 - \psi_\lambda = 0.404$), both of which are better than $\Psi(\mathbf{Y}^*) = 0.533$ (*i.e.*, $\Phi^* = 1 - \Psi(\mathbf{Y}^*) = 0.467$) 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 [41] 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 $\mathbf{R}^\top\mathbf{R} = \mathbf{I}_k$. 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 the objective in Eq. (10), and thus, produces sub-par result quality, as revealed by experiments (Table 14).

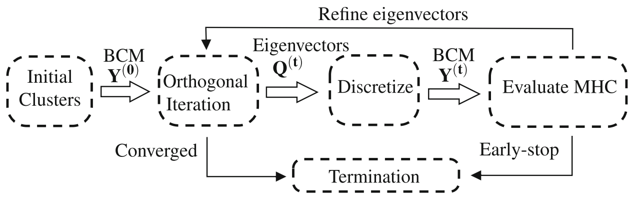

**Fig. 3** Overview of AHCKA

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 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* [42] 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 [41]. 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 (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 Sect. 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 Sect. 5.3.

**Algorithm 1:** AHCKA

```
Input: Hypergraph $\mathcal{H}$, KNN transition matrix $\mathbf{P}_K$, the number of
       clusters $k$, diagonal matrix $\mathbf{B}$, constant $\alpha$, error threshold
       $\epsilon_Q$, the numbers of iterations $T_a, \gamma$, an integer $\tau$, 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. (12);
5      $\mathbf{Q}^{(t)}, \mathbf{R}^{(t)} \leftarrow$ QR($\mathbf{Z}^{(t)}$) ;
6      if $t$ mod $\tau = 0$ then
7          $\mathbf{Y}^{(t)} \leftarrow$ Discretize($\mathbf{Q}^{(t)}$) ;
8          $\Phi(\mathbf{Y}^{(t)}) \leftarrow$ 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. (15) or Eq. (16) holds then break;
11 return $\mathbf{Y}$;
```

## 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. (2), the random walk stopping probability $\alpha$, an error threshold $\epsilon_Q$, the numbers $\gamma$, $T_a$ of iterations, an integer $\tau$, 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. (8)) 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 updates the approximate $k+1$ leading eigenvectors of $\mathbf{P}$ as $\mathbf{Q}^{(t)}$ by the formula below (Lines 4–5):

$$\mathbf{Q}^{(t)}\mathbf{R}^{(t)} = \mathbf{Z}^{(t)} = \mathbf{P}\mathbf{Q}^{(t-1)}, \tag{11}$$

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 $\mathbf{P}$ [42]. Note that the direct computation of $\mathbf{Z}^{(t)} = \mathbf{P}\mathbf{Q}^{t-1}$ requires constructing $\mathbf{P}$ explicitly as per Eq. (4), which incurs an exorbitant amount of time and space (up to $O(n^2)$ in the worst case). To mitigate this, we decouple and reorder the matrix multiplication as in Eq. (12).

$$\mathbf{Z}^{(t)} = (\mathbf{I} - \mathbf{B}) \cdot \mathbf{P}_V \cdot \left(\mathbf{P}_E \mathbf{Q}^{(t-1)}\right) + \mathbf{B}\mathbf{P}_K \cdot \mathbf{Q}^{(t-1)}, \tag{12}$$

$$\text{where } \mathbf{P}_V = \mathbf{D}_V^{-1}\mathbf{H}^\top,\ \mathbf{P}_E = \mathbf{D}_E^{-1}\mathbf{H} \tag{13}$$

$\mathbf{P}_V$ and $\mathbf{P}_E$ 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 Sect. 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 $O(nk \cdot (\bar{\delta} + K))$.

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 [41]. Notice that we conduct this conversion every other $\tau$ 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 $\gamma$ as input to calculate the MHC $\phi_t$ of the current BCM matrix $\mathbf{Y}^{(t)}$. To avoid the materialization of $\mathbf{S}$ required in Eqs. (9) and (10), Algorithm 2 computes $\phi_t$ in an iterative manner by reordering the matrix multiplications (Lines 2–3 in Algorithm 2). More precisely, at the $\ell$-th iteration, it obtains the intermediate result $\mathbf{F}^{(\ell)}$ via the following equation:

**Algorithm 2:** CalMHC

```
Input: $\mathbf{Y}^{(t)}, \mathbf{P}_V, \mathbf{P}_E, \mathbf{P}_K, \mathbf{B}, \gamma, \alpha$
Output: MHC $\phi_t$
1  $\widehat{\mathbf{Y}}^{(t)} \leftarrow h(\mathbf{Y}^{(t)})$;  $\mathbf{F}^{(0)} \leftarrow \alpha\widehat{\mathbf{Y}}^{(t)}$;
2  for $\ell \leftarrow 1, 2, \ldots \gamma$ do
3      Compute $\mathbf{F}^{(\ell)}$ according to Eq. (14);
4  $\phi_t \leftarrow 1 - \frac{1}{k} trace(\widehat{\mathbf{Y}}^{(t)\top}\mathbf{F}^{(\gamma)})$;
5  return $\phi_t$ ;
```

$$\mathbf{F}^{(\ell)} = (1-\alpha)\left((\mathbf{I} - \mathbf{B}) \cdot \mathbf{P}_V \cdot \left(\mathbf{P}_E \mathbf{F}^{(\ell-1)}\right) + \mathbf{B}\mathbf{P}_K \cdot \mathbf{F}^{(\ell-1)}\right) + \mathbf{F}^{(0)}. \tag{14}$$

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

Once the convergence criterion of $\mathbf{Q}^{(t)}$ (Eq. (15)) is satisfied, or the early termination condition (Eq. (16)) holds, AHCKA ceases the iterative process and returns the BCM matrix $\mathbf{Y}$ with the lowest MHC (Lines 9–11 in Algorithm 1).

$$||\mathbf{Q}^{(t)} - \mathbf{Q}^{(t-1)}|| < \epsilon_Q \tag{15}$$

$$\phi_{t-2\tau} < \phi_{t-\tau} < \phi_t \tag{16}$$

Otherwise, AHCKA proceeds to the next orthogonal iteration. The rationale for the early termination condition in Eq. (16) is that, in practice, successive increases in $\phi_t$ indicate that clusters with desirable MHC objective have been attained.

## 5.2 Greedy initialization of BCM

Akin to many optimization problems, AHCKA requires many iterations to achieve convergence when $\mathbf{Y}^{(0)}$ is randomly initialized. To tackle this issue, we propose a greedy initialization technique, InitBCM, whereby we can immediately

**Algorithm 3:** `InitBCM`

```
Input: Hypergraph $\mathcal{H}$, matrices $\mathbf{P}_V$, $\mathbf{P}_E$, integer $k$, constant $\alpha$,
       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 $\boldsymbol{\Pi}_c^{(0)} \leftarrow \alpha \mathbf{Z}_0$;
5  for $t \leftarrow 1, 2, \ldots T_i$ do
6      Compute $\boldsymbol{\Pi}_c^{(t)}$ according to Eq. (17);
7  for $v_j \in \mathcal{V}$ do
8      Calculate $g(v_j)$ according to Eq. (18);
9      $\mathbf{Y}^{(0)}[j, g(v_j)] \leftarrow 1$;
10 return $\mathbf{Y}^{(0)}$ ;
```

gain a passable BCM matrix $\mathbf{Y}^{(0)}$ and expedite the convergence, as demonstrated by our experiments in Sect. 8.4.

The rationale of `InitBCM` is that most nodes tend to cluster together around a number of center nodes [43]. Therefore, we can first pick a set $\mathcal{V}_c$ of top influential nodes w.r.t. 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. (13), the number $k$ of clusters, random walk stopping probability $\alpha$, 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 5–6). Specifically, at $t$-th iteration, we compute approximate RWR $\boldsymbol{\Pi}_c^{(t)}$ (Line 6):

$$\boldsymbol{\Pi}_c^{(t)} = (1-\alpha)\left(\boldsymbol{\Pi}_c^{(t-1)}\mathbf{P}_V\right)\cdot \mathbf{P}_E + \boldsymbol{\Pi}_0, \tag{17}$$

where $\boldsymbol{\Pi}_0 = \alpha \mathbf{Z}_0$ (Line 4). Note that we reorder the matrix multiplications as in Eq. (17) so as to bypass the materialization of the $n \times n$ matrix $\mathbf{P}_V \mathbf{P}_E$. After obtaining $\boldsymbol{\Pi}_c^{(T_i)}$, `InitBCM` assigns the node $\mathcal{V}_c[g(v_j)]$ as the cluster center to each node $v_j$ in $\mathcal{H}$ as per Eq. (18) (Lines 7–9).

$$g(v_j) = \arg\max_{1\le l\le k} \boldsymbol{\Pi}_c^{(T_i)}[l, j], \tag{18}$$

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

## 5.3 Complexity

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. (12). 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 $O((n\overline{\delta}+nK)\cdot k)$ [44]. Analogously, according to Eqs. (14), and (17), 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 $T_a$, $\gamma$, 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 `Discretize` [41] 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 $T_a$, $T_i$, $\gamma$, $k$, 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 The `ANCKA` framework

In this section, we generalize `AHCKA` that is for AHC to a versatile framework `ANCKA` to process all of AHC, AGC, and AMGC, formulated in Sect. 2. `ANCKA` aims to efficiently find high-quality clusters on various types of network $\mathcal{N}$.

As mentioned, the proposed KNN augmentation in Sect. 3.1 is orthogonal to the high-order nature of hypergraph, and therefore, we can apply the KNN augmentation to input attributed network $\mathcal{N}$ that can be an attributed hypergraph $\mathcal{H}$, graph $\mathcal{G}$, and multiplex graph $\mathcal{G}_M$.

Recall that, in Fig. 2, we have empirically shown that nodes with higher attribute similarity are more likely to appear in the same cluster of a hypergraph $\mathcal{H}$. This also holds for attributed graphs and attributed multiplex graphs. Figures 4 and 5 illustrate the AAS and RCC on the attributed graph Citeseer-DG and the attributed multiplex graph ACM, with binary keyword vectors as node attributes. On both datasets, nodes with higher attribute similarity (i.e., higher AAS with smaller $K$) are more likely to be in the same cluster (i.e., higher RCC). Moreover, above a certain $K$ value, there is no significant difference between the RCC of two random nodes and that of two nodes $v_i$ and $v_j$ such that $v_j$ is the K-nearest neighbor of $v_i$. Based on these observations, it is viable to extend KNN augmentation in Sect. 3.1 to an attributed network $\mathcal{N}$ with $n$ nodes and attribute matrix

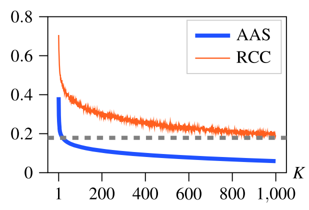

**Fig. 4** AAS and RCC on Citeseer-DG

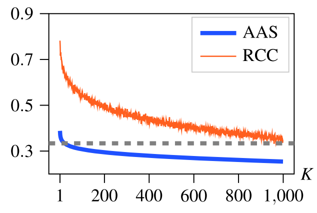

**Fig. 5** AAS and RCC on ACM

$\mathbf{X} \in \mathbb{R}^{n\times d}$, by building a KNN augmentation graph $\mathcal{G}_K$ via Eq. (1).

Then we obtain an augmented network $\mathcal{N}_A$ with topology $\mathcal{N}_O$ and KNN graph $\mathcal{G}_K$, where $\mathcal{N}_O$ is $(\mathcal{V}, \mathcal{E})$ when $\mathcal{N}$ is an attributed hypergraph $\mathcal{H}$ or $(\mathcal{V}, \mathcal{E}_G)$ for graph $\mathcal{G}$, and $\mathcal{N}_O$ is $(\mathcal{V}, \mathcal{E}_1, \ldots, \mathcal{E}_L)$ when $\mathcal{N}$ is an attributed multiplex graph $\mathcal{G}_M$.

### 6.1 Generalized $(\alpha, \beta, \gamma)$-random walk

For the augmented network $\mathcal{N}_A = (\mathcal{N}_O, \mathcal{G}_K)$, define $\mathbf{P}_N$ and $\mathbf{P}_K$ as the random walk transition matrices of $\mathcal{N}_O$ and $\mathcal{G}_K$ respectively. The generalized $(\alpha, \beta, \gamma)$-random walk on $\mathcal{N}_A$ is an RWR process over the augmented network $\mathcal{N}_A$, similar to the case of attributed hypergraphs in AHCKA. The difference from Definition 1 is that when the random walk navigates to another node, with probability $1 - \beta_i$, an out-neighbor is drawn from the distribution of $\mathbf{P}_N$ instead of incident hyperedges. This generalized random walk can also be characterized by the probability in Eq. (3), with transition matrix $\mathbf{P}$ given as follows.

$$\mathbf{P} = (\mathbf{I} - \mathbf{B}) \cdot \mathbf{P}_N + \mathbf{B} \cdot \mathbf{P}_K. \tag{19}$$

We now formulate $\mathbf{P}_N$ for different types of networks, including attributed hypergraphs as one special case.

**Attributed Hypergraph.** $\mathcal{H}$ When $\mathcal{N}_O$ is a hypergraph with hyperedge incidence matrix $\mathbf{H}$, based on Eq. (4), $\mathbf{P}_N$ is shown below. $\mathbf{P}_N$ considers the transition probability $\mathbf{P}_V$ from a node to its incident hyperedges and the transition probability $\mathbf{P}_E$ from each hyperedge to nodes connected by the hyperedge.

$$\mathbf{P}_N = \mathbf{P}_V \mathbf{P}_E, \text{ where } \mathbf{P}_V = \mathbf{D}_V^{-1}\mathbf{H}^\top \text{ and } \mathbf{P}_E = \mathbf{D}_E^{-1}\mathbf{H}. \tag{20}$$

**Attributed Graph** $\mathcal{G}$. When $\mathcal{N}_O$ is an undirected graph, we can acquire the transition matrix $\mathbf{P}_N$ in Eq. (21). If $\mathcal{N}_O$ is directed, we introduce a reversed edge for each edge and consider bidirectional connections between nodes to get $\mathbf{A}$, $\mathbf{D}$, and subsequently $\mathbf{P}_N$.

$$\mathbf{P}_N = \mathbf{D}^{-1}\mathbf{A}, \tag{21}$$

where $\mathbf{A}$ is the adjacency matrix and $\mathbf{D}$ is the degree matrix.

**Attributed Multiplex Graph.** $\mathcal{G}_M$ When $\mathcal{N}_O$ is a multiplex graph comprising $L$ layers with the same node set $\mathcal{V}$, the $l$-th layer has its own edge set $\mathcal{E}_l$ representing a unique type of connections. The overall goal of the clustering task is to make cluster assignments that capture the collective structure of the multiplex graph, transcending the differences across layers. To achieve this, intuitively, we treat every layer equally and compute $\mathbf{P}_N$ as in Eq. (22), while layer weighting is left as future work [27]. Given the degree matrix $\mathbf{D}_l$ and adjacency matrix $\mathbf{A}_l$ of every $l$-th layer, we get the layer's random walk transition matrix $\mathbf{D}_l^{-1}\mathbf{A}_l$, and then compute $\mathbf{P}_N$ of the multiplex graph by averaging the layer-specific transition matrices. Consequently, from the current node $v$, a random walk has $1/L$ probability of selecting each layer $\mathcal{G}_l$, and then within this chosen layer, the next node to visit is picked uniformly at random from the out-neighbors of $v$ in $\mathcal{G}_l$.

$$\mathbf{P}_N = \frac{1}{L}\sum_{l=1}^{L} \mathbf{D}_l^{-1}\mathbf{A}_l, \tag{22}$$

where $\mathbf{D}_l$ and $\mathbf{A}_l$ are the degree matrix and adjacency matrix of the $l$-th layer.

### 6.2 ANCKA algorithm

With the random walk transition matrix $\mathbf{P}$ formulated above for various types of attributed networks $\mathcal{N}$, Eq. (3) can be reused to calculate $\mathbf{S}[i, j]$, the probability of a generalized $(\alpha, \beta, \gamma)$-random walk from $v_i$ stopping at $v_j$ in the end. The objective function in Sect. 3.3 is naturally extended to ANCKA. Consequently, our theoretical analysis in Sect. 4 remains valid for ANCKA over attributed networks that can be hypergraphs, graphs, and multiplex graphs.

The pseudo-code of ANCKA is outlined in Algorithm 4. At Line 1, it obtains transition matrix $\mathbf{P}_K$ for attribute KNN augmentation. Then as a framework supporting various attributed networks, ANCKA is a generalization of Algorithms 1–3 with transition matrix $\mathbf{P}_N$ computed depending on the network type at Line 2. $\mathbf{P}_N$ is then used throughout the algorithm as a part of the generalized $(\alpha, \beta, \gamma)$-random walk. The greedy initialization of clusters in Lines 3–11 resembles the procedure in InitBCM with the corresponding $\mathbf{P}_N$ for RWR simulation. Since ANCKA needs to pick $k$ nodes in $\mathcal{N}$

**Algorithm 4:** ANCKA

```
Input: Attributed network $\mathcal{N}$ with KNN augmented graph $\mathcal{G}_K$,
       the number of clusters $k$, diagonal matrix $\mathbf{B}$, constant $\alpha$,
       error threshold $\epsilon_Q$, the numbers of iterations $T_a, \gamma, T_i$, an
       integer $\tau$.
Output: BCM matrix $\mathbf{Y}$
1  $\mathbf{P}_K \leftarrow \mathbf{D}_K^{-1}\mathbf{A}_K$;
2  Get $\mathbf{P}_N$ by Eqs. (20), (21), or (22), depending on the type of $\mathcal{N}$;
3  $\mathcal{V}_c \leftarrow$ sorted indices of $k$ nodes in $\mathcal{N}$ with $k$ largest degrees;
4  Initialize $\mathbf{Z}_0 \leftarrow \mathbf{0}^{k\times n}$;
5  for $j \leftarrow 1$ to $k$ do $\mathbf{Z}_0[j, \mathcal{V}_c[j]] \leftarrow 1$ ;
6  Initialize $\boldsymbol{\Pi}_c^{(0)} \leftarrow \alpha\mathbf{Z}_0$;
7  for $t \leftarrow 1, 2, \dots T_i$ do
8      $\boldsymbol{\Pi}_c^{(t)} \leftarrow (1-\alpha)\boldsymbol{\Pi}_c^{(t-1)}\mathbf{P}_N + \boldsymbol{\Pi}_c^{(0)}$;
9  for $v_j \in \mathcal{V}$ do
10     $g(v_j) \leftarrow \arg\max_{1\le l\le k} \boldsymbol{\Pi}_c^{(T_i)}[l, j]$ ;
11     $\mathbf{Y}^{(0)}[j, g(v_j)] \leftarrow 1$;
12 $\mathbf{Y} \leftarrow \mathbf{Y}^{(0)}$, $\widehat{\mathbf{Y}}^{(0)} \leftarrow h(\mathbf{Y}^{(0)})$;
13 $\mathbf{Q}^{(0)} \leftarrow \frac{1}{\sqrt{n}} \cdot \mathbf{1}|\widehat{\mathbf{Y}}^{(0)}$ ;
14 for $t \leftarrow 1, 2, \cdots, T_a$ do
15     $\mathbf{Z}^{(t)} \leftarrow (\mathbf{I}-\mathbf{B})\mathbf{P}_N \cdot \mathbf{Q}^{(t-1)} + \mathbf{B}\mathbf{P}_K \cdot \mathbf{Q}^{(t-1)}$;
16     $\mathbf{Q}^{(t)}, \mathbf{R}^{(t)} \leftarrow$ QR($\mathbf{Z}^{(t)}$) ;
17     if $t$ mod $\tau = 0$ then
18         $\mathbf{Y}^{(t)} \leftarrow$ Discretize($\mathbf{Q}^{(t)}$) ;
19         $\widehat{\mathbf{Y}}^{(t)} \leftarrow h(\mathbf{Y}^{(t)})$; $\mathbf{F}^{(0)} \leftarrow \alpha\widehat{\mathbf{Y}}^{(t)}$;
20         for $\ell \leftarrow 1, 2, \dots \gamma$ do
21             Compute $\mathbf{F}^{(\ell)}$ according to Eq. (23)
22         $\Phi(\mathbf{Y}^{(t)}) \leftarrow 1 - \frac{1}{k} trace(\widehat{\mathbf{Y}}^{(t)\top}\mathbf{F}^{(\gamma)})$;
23         if $\Phi(\mathbf{Y}^{(t)}) < \Phi(\mathbf{Y})$ then $\mathbf{Y} \leftarrow \mathbf{Y}^{(t)}$;
24         if Eq. (15) or Eq. (16) holds then break;
25 return $\mathbf{Y}$;
```

with the largest degrees as tentative cluster centers at Line 3 when $\mathcal{N}$ is an attributed multiplex graph, we rank the nodes by their summed degrees across all layers.

Lines 12–24 describe the main clustering process of ANCKA, which extends the hypergraph-specific Algorithms 1 and 2 with modifications to support attributed graphs and multiplex graphs. First, in orthogonal iterations, calculating $\mathbf{Z}^{(t)}$ is dependent on the type of $\mathcal{N}$. Second, the MHC objective for general networks stems from the analysis in Sect. 4, while the formulation with $\mathbf{P}_N$ is slightly different. In particular, to get MHC $\phi_t$ without materializing the dense matrix $\mathbf{S}$ in Eq. (10) that is expensive to compute, we iteratively obtain $\phi_t$ via the intermediate matrix $\mathbf{F}^{(\ell)}$ in Eq. (23) at Line 21.

$$\mathbf{F}^{(\ell)} = (1-\alpha)((\mathbf{I}-\mathbf{B})\mathbf{P}_N\mathbf{F}^{(\ell-1)} + \mathbf{B}\mathbf{P}_K\mathbf{F}^{(\ell-1)}) + \mathbf{F}^{(0)}, \tag{23}$$

where $\mathbf{P}_N$ is Eqs. (20), (21), or (22), depending on the type of $\mathcal{N}$. Finally, ANCKA adopts the early stopping criteria in Line 24 and returns the clusters with the lowest MHC obtained.

**Complexity.** When $\mathcal{N}$ is an attributed graph, constructing transition matrix $\mathbf{P}_N$ takes $O(n\overline{\delta})$ time, where $\overline{\delta}$ is the average node degree. For a multiplex network $\mathcal{N}$ with $L$ layers, the previous results are still valid when $L$ is regarded as constant, as $\mathbf{P}_N$ is aggregated from the transition matrices of all simple graph layers. Given that the number of nonzero entries in $\mathbf{P}_N$ is subject to $O(n\overline{\delta})$, ANCKA (Algorithm 4) has the same complexity as Algorithm 1. According to our analysis in Sect. 5.3, the time complexity of ANCKA is $O(kn(\overline{\delta}+K+k))$ while its space complexity is $O(n(\overline{\delta}+K+k))$. Since $k$ and $K$ can be viewed as constants, ANCKA has space and time complexity of $O(n\overline{\delta})$.

## 7 GPU-accelerated ANCKA-GPU

On large attributed networks, e.g., Amazon and MAG-PM hypergraphs, each with more than 2 million nodes, as reported in Table 7, AHCKA with 16 CPU threads still needs 1286s and 1372s respectively for clustering, despite its superior efficiency compared with baselines. Moreover, AHCKA does not exhibit acceleration proportional to increased CPU threads. As shown in Fig. 6, when the number of CPU threads is raised from 1 to 32, the time drops from around 3000 to 1200 s, with a speedup of merely 2.5 (Amazon) or 2.7 (MAG-PM). In particular, increasing the number of threads from 16 to 32 provides rather limited acceleration (less than 10%).

To overcome the limitation of CPU parallelization, we resort to the massive parallel processing power of GPUs (graphical processing unit) and develop ANCKA-GPU to boost efficiency, with about one order of magnitude speedup on large networks with millions of nodes in experiments. For example, ANCKA-GPU only needs 120 s on an MAG-PM dataset, over 10 times faster than the 1372s of ANCKA. Compared to CPUs, the design of GPUs enables them to leverage numerous threads to handle data processing simultaneously, which is beneficial for vector and matrix operations at scale. Please see [45] for details on GPU computing.

As shown in Fig. 15 of Sect. 8.5 for runtime analysis, the major time-consuming components of ANCKA include invoking Discretize (Line 18 in Algorithm 4), the construction of KNN graph $\mathcal{G}_K$, and expensive matrix operations in orthogonal iterations, greedy initialization and MHC evaluation. With the CuPy library, matrix operations throughout Algorithm 4 can be done on GPUs more efficiently. In the following, we elaborate on the GPU-based discretization and $\mathcal{G}_K$ construction techniques adopted in ANCKA-GPU.

**GPU-based Discretization Discretize-GPU.** ANCKA uses the off-the-shelf Discretize approach [41] to compute discrete cluster labels $\mathbf{Y}$ from real-valued eigenvectors $\mathbf{Q}$, which could cost substantial time on large datasets. Here, we develop a CUDA kernel Discretize-GPU for effi-

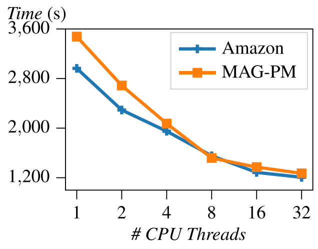

**Fig. 6** Runtime of AHCKA with CPU parallelization

ciency. In what follows, we first explain how the discretization algorithm improves the optimization objective in Definition 2, and then present the design of `Discretize-GPU` in Algorithm 5.

Given an eigenvector matrix $\mathbf{Q}$ with its row-normalized matrix $\tilde{\mathbf{Q}}$, discretization is aimed to find a discrete solution $\mathbf{Y}_{opt}$ that minimizes the objective in Definition 2.

**Definition 2** (Discretization [41]) The solution to the following optimization problem is the optimal discrete $\mathbf{Y}_{opt}$.

$$\mathbf{Y}_{opt} = \underset{\mathbf{Y}}{\operatorname{argmin}} \|\mathbf{Y} - \tilde{\mathbf{Q}}\mathbf{R}\|_F^2$$
$$\text{s.t. } \mathbf{Y} \in \{0,1\}^{n\times k},\ \mathbf{Y}\mathbf{1}_k = \mathbf{1}_n,\ \mathbf{R} \in \mathbb{R}^{k\times k},\ \mathbf{R}^\top\mathbf{R} = \mathbf{I}_k,$$

where $\tilde{\mathbf{Q}}$ is the row-normalized matrix of an eigenvector matrix $\mathbf{Q}$, $\mathbf{R}$ is a rotation matrix, and $\|\mathbf{M}\|_F$ denotes the Frobenius norm of matrix $\mathbf{M}$.

The `Discretize` approach finds a nearly global optimal solution by alternately updating one of $\mathbf{Y}$ and $\mathbf{R}$ while keeping the other fixed. With $\mathbf{R}$ fixed, $\mathbf{Y}[i,l]$ is updated to

$$\mathbf{Y}[i,g] = \begin{cases} 1, & \text{if } g = \arg\max_{1\le j\le k}(\tilde{\mathbf{Q}}\mathbf{R})[i,j] \\ 0, & \text{otherwise.} \end{cases} \tag{24}$$

With $\mathbf{Y}$ fixed, $\tilde{\mathbf{Y}}$ is the column-normalized matrix of $\mathbf{Y}$, and $\mathbf{R}$ can be updated as follows with SVD decomposition.

$$\mathbf{R} = \mathbf{V}\mathbf{U}^\top, \text{ where } \mathbf{U}\boldsymbol{\Omega}\mathbf{V}^\top \text{ is an SVD of } \tilde{\mathbf{Y}}^\top\tilde{\mathbf{Q}}. \tag{25}$$

The iterative process can terminate early when an objective value *obj* based on $\boldsymbol{\Omega}$ converges, i.e., its change over the last iteration is within machine precision. This objective is calculated as $obj = n - 2 \times trace(\boldsymbol{\Omega})$ [41].

We implement the CUDA kernel `Discretize-GPU` in Algorithm 5 to perform the process above to obtain $\mathbf{Y}$. In details, `Discretize-GPU` leverages the grid-block-thread hierarchy of GPU to assign threads to handle $n \times k$ matrices, including $\mathbf{Q}$ and $\mathbf{Y}$. Each row in such a matrix is processed by a block of threads, identified by a block id $bid$; each of the $k$ elements in the row is handled by a thread $tid$ in the block. Consequently, given a matrix $\mathbf{Q}$, we can use $\mathbf{Q}[bid, tid]$ to represent that the corresponding element in $\mathbf{Q}$ is handled by

**Algorithm 5:** `Discretize-GPU`

```
   Input: eigenvector matrix $\mathbf{Q}$
   Output: Intermediate BCM matrix $\mathbf{Y}$
1  Parallel for $i \leftarrow 1, 2, \cdots, n$ do
2      $\tilde{\mathbf{Q}}[i] \leftarrow \frac{\mathbf{Q}[i]}{\|\mathbf{Q}[i]\|_2}$ ;
3  $\mathbf{R} \leftarrow \mathbf{I}_k$ ;
4  while $iter \leftarrow 1, 2, \cdots, max\_iter$ do
5      Update $\mathbf{Y}$ by Eq. (24) via argmax kernel on GPU;
6      Parallel for $j \leftarrow 1, 2, \cdots, k$ do
7          $col\_sum[j] \leftarrow \sum_{i=1}^{n} \mathbf{Y}[i, j]$
8      Parallel for each $tid < k$ in blocks do
9          $\tilde{\mathbf{Y}}[bid, tid] \leftarrow \frac{\mathbf{Y}[bid,tid]}{col\_sum[tid]}$;
10     $\mathbf{U}, \boldsymbol{\Omega}, \mathbf{V}^\top \leftarrow$ SVD_GPU($\tilde{\mathbf{Y}}^\top\tilde{\mathbf{Q}}$) ;
11     $\mathbf{R} \leftarrow \mathbf{V}\mathbf{U}^\top$ on GPU;
12     if Objective value obj does not change then break;
13 return $\mathbf{Y}$;
```

the $tid$-th thread in block $bid$ on a GPU. Parallel row normalization is performed at Lines 1–2 to get $\tilde{\mathbf{Q}}$. After initializing $\mathbf{R}$ as a $k \times k$ identity matrix (Line 3), we alternately update $\tilde{\mathbf{Y}}$ and $\mathbf{R}$ for at most $max\_iter$ iterations (Lines 4–12) and terminate early when the objective value *obj* does not change over the current iteration at Line 12. Within an iteration, we first update $\mathbf{Y}$ at Line 5, then perform column normalization to get $\tilde{\mathbf{Y}}$ (Lines 6–9), and then perform SVD on GPU over $\tilde{\mathbf{Y}}^\top\tilde{\mathbf{Q}}$ to get $\mathbf{U}$ and $\mathbf{V}$ at Line 10, which helps to update $\mathbf{R}$ at Line 11. Finally, $\mathbf{Y}$ is returned at Line 13.

**KNN construction.** An $n \times d$ attributed matrix $\mathbf{X}$ requires KNN search on its rows to construct the augmented graph $\mathcal{G}_K$ and thus the transition matrix $\mathbf{P}_K$. For this purpose, we adopt Faiss [30], a GPU-compatible similarity search library. In Algorithm 6 for $\mathcal{G}_K$ construction, we first normalize all rows in $\mathbf{X}$ at Lines 1–2 to facilitate the computation of cosine similarity between row vectors. Faiss supports various indexes for KNN computation, and the index type suitable for ANCKA is determined based on the input data volume. For small or medium datasets where the number of nodes $|\mathcal{V}|$ is below 100,000, since the time cost for exact similarity search is affordable, we choose the flat index with a plain encoding of each row vector in $\mathbf{X}$, to achieve exact KNN computation (Lines 3–4). Otherwise, we turn to approximate nearest neighbor search on large datasets with the IVFPQ index that combines the inverted file index (IVF) with the product quantization (PQ) technique at Line 6. In particular, IVF index narrows down the search to closely relevant partitions that contain the nearest neighbors at a high probability, while PQ produces memory-efficient encoding of attribute vectors. Faiss on GPU is invoked to get the KNN of each row in $\mathbf{X}$, and $\mathbf{A}_K$ is obtained by Eq. (1) at Lines 7–8. Then, the degree matrix $\mathbf{D}_K$ and transition matrix $\mathbf{P}_K$ are computed on GPU (Lines 9-10) and returned at Line 11.

**Algorithm 6:** GPU-based $\mathcal{G}_K$ construction

```
   Input: Network $\mathcal{N}$, attribute matrix $\mathbf{X}$, parameter $K$.
   Output: KNN transition matrix $\mathbf{P}_K$
1  Parallel for $i \leftarrow 1, 2, \cdots, n$ do
2      $\mathbf{X}[i] \leftarrow \frac{\mathbf{X}[i]}{\|\mathbf{X}[i]\|_2}$;
3  if $|\mathcal{V}| < 100,000$ then
4      $index \leftarrow$ FlatIndex ($\mathbf{X}$) ;
5  else
6      $index \leftarrow$ IVFPQIndex ($\mathbf{X}$) ;
7  Invoke Faiss on GPU to get the KNN of each row in $\mathbf{X}$ ;
8  Get $\mathbf{A}_K$ by Eq. (1) on GPU ;
9  $\mathbf{D}_K \leftarrow$ Diag($\mathbf{A}_K \mathbf{1}_n$) on GPU ;
10 $\mathbf{P}_K \leftarrow \mathbf{D}_K^{-1}\mathbf{A}_K$ on GPU;
11 return $\mathbf{P}_K$;
```

## 8 Experiments

We evaluate the proposed ANCKA and competitors in terms of clustering quality and efficiency. We also evaluate the performance of ANCKA-GPU on all clustering tasks. In experiments, we uniformly refer to our method as ANCKA while making it clear in the context whether ANCKA is for AHC (i.e., AHCKA), AGC, or AMGC. All the experiments are conducted on a Linux machine powered by Intel Xeon(R) Gold 6226R CPUs, 384GB RAM, and NVIDIA RTX 3090 GPU. A maximum of 16 CPU threads are available if not otherwise stated. The code is at https://github.com/gongyguo/ANCKA.

### 8.1 Experimental setup

#### 8.1.1 Datasets

Table 1 provides the statistics of 17 real-world attributed networks used in experiments, including attributed hypergraphs (HG), undirected graphs (UG), directed graphs (DG), and multiplex graphs (MG). $|\mathcal{V}|$ and $|\mathcal{E}|$ are the number of nodes and edges (or hyperedges), respectively, $d$ is the attribute dimension and $k$ is the number of ground-truth clusters.

We gather 8 attributed hypergraph datasets. Query dataset [5] is a Web query hypergraph, where nodes represent queries and are connected by hyperedges representing query sessions, and nodes are associated with attributes of keyword embeddings and associated webpages. Cora-CA, Cora-CC, Citeseer, and DBLP are four benchmark datasets used in prior work [46]. All of them are originally collected from academic databases, where each node represents a publication, node attributes are binary word vectors of abstract, 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 [47] consists of messages taken from Usenet 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 attribute. Amazon dataset is constructed based on the 5-core subset of Amazon reviews dataset [48], 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 [49], 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).

In Table 1, we also consider 6 attributed graphs, which are commonly used for AGC [13, 27, 50, 51]. Cora, Citeseer-UG, Wiki, and Amazon2M are undirected, while Citeseer-DG and TWeibo are directed. TWeibo [13] and Amazon2M [51] are two large-scale attributed graphs. TWeibo is a social network where each node represents a user, and the directed edges represent relationships between users. Amazon2M is constructed based on the co-purchasing networks of products on Amazon. Cora, Citeseer-UG, and Citeseer-DG are citation networks where nodes represent publications, a pair of nodes are connected if one cites the other, and nodes are associated with binary word vectors as features. Wiki is a webpage network where each edge in the graph indicates that one webpage is linked to the other, while the node attributes are TF-IDF feature vectors.

Moreover, three attributed multiplex graphs, namely ACM, IMDB, and DBLP-MG, are considered for AMGC [22–24]. ACM is an academic publication network comprising co-author and co-subject graph layers, as well as bag-of-words attributes of keywords. IMDB is a movie network with plot text embeddings as attributes and two graph layers representing the co-director (directed by the same director) and co-actor (starring the same actor) relations, respectively. DBLP-MG is a researcher network including publication keyword vectors as attributes and three graph layers: co-author, co-conference (publishing at the same conference), and co-term (sharing common key terms). ACM and DBLP-MG have research areas labeled as ground truth clusters, while IMDB is labeled by movie genres.

#### 8.1.2 Competitors and parameter settings

The 19 competitors for AHC are summarized as follows:

- 3 plain hypergraph clustering methods including HNCut [52], HyperAdj [53], and KaHyPar [54];
- the extended AHC versions of the 3 methods above (dubbed as ATHNCut, ATHyperAdj, and ATKaHyPar), which work on an augmented hypergraph with attribute-

**Table 1** Dataset statistics

| Task | Dataset | Type | $\|\mathcal{V}\|$ | $\|\mathcal{E}\|$ | $d$ | $k$ |
|---|---|---|---|---|---|---|
| AHC | Query | HG | 481 | 15, 762 | 426 | 6 |
| | Cora-CA | HG | 2708 | 1072 | 1433 | 7 |
| | Cora-CC | HG | 2708 | 1579 | 1433 | 7 |
| | Citeseer | HG | 3312 | 1079 | 3703 | 6 |
| | 20News | HG | 16, 242 | 100 | 100 | 4 |
| | DBLP | HG | 41, 302 | 22, 363 | 1425 | 6 |
| | Amazon | HG | 2, 268, 083 | 4, 285, 295 | 1000 | 15 |
| | MAG-PM | HG | 2, 353, 996 | 1, 082, 711 | 1000 | 22 |
| AGC | Cora | UG | 2708 | 5429 | 1433 | 7 |
| | Citeseer-UG | UG | 3327 | 4732 | 3703 | 6 |
| | Wiki | UG | 2405 | 17, 981 | 4973 | 17 |
| | Citeseer-DG | DG | 3312 | 4715 | 3703 | 6 |
| | TWeibo | DG | 2320, 895 | 50, 655, 143 | 1657 | 8 |
| | Amazon2M | UG | 2, 449, 029 | 61, 859, 140 | 100 | 47 |
| AMGC | ACM | MG | 3025 | 29, 281 | 1870 | 3 |
| | | | | 2, 210, 761 | | |
| | IMDB | MG | 3550 | 13, 788 | 2000 | 3 |
| | | | | 66, 428 | | |
| | DBLP-MG | MG | 4057 | 11, 113 | 334 | 4 |
| | | | | 5, 000, 495 | | |
| | | | | 7, 043, 571 | | |

KNN hyperedges of all nodes merged into the input hypergraph;
- ATMetis that applies the traditional graph clustering algorithm Metis [55] over a graph constructed by clique expansion of the input hypergraph and attribute KNN graph augmentation; Infomap [56], Louvain [57], k-MQI and k-Nibble (extended from MQI [58] and PageRank-Nibble [59] for $k$-way clustering via $k-1$ consecutive bisections as described in technical report [40]) on the same KNN-augmented clique-expansion graph;
- 3 AHC algorithms including the recent GRAC [11] and NMF-based approaches (GNMF [11, 60] and JNMF [1]);
- ACMin-C and ACMin-S, obtained by applying an attributed graph clustering method ACMin [13] over the graphs reduced from hypergraphs by clique expansion and star expansion, respectively; probabilistic model CESNA [3] with clique-expansion;
- k-means and HAC (hierarchical agglomerative clustering [61]) algorithms applied to the node attribute matrix.

To evaluate the ANCKA framework, we compare 16 competitors for AGC, including k-means, HAC and the following:

- 6 AGC approaches including NMF-based algorithm GNMF [60], graph convolution algorithm AGCGCN [50], probabilistic model CESNA [3], spectral clustering on fine-grained graphs method FGC [62], attributed random walk approach ACMin [13], and the clustering framework GRACE [27] generalized from GRAC.
- NCut [37] and Metis [55] that are conventional graph clustering methods applied to the input graph;
- ATNCut and ATMetis that are NCut and Metis applied to the augmented graph with attribute KNN; Infomap [56], Louvain [57], k-MQI [58] and k-Nibble [59] on the augmented graph with attribute KNN.

We compare ANCKA with 16 competitors for AMGC task, including k-means, HAC and the following:

- 5 AMGC methods: a multi-view graph auto-encoder model O2MAC [25], HDMI [24] that learns node embeddings via higher-order mutual information loss, MCGC [22] and MAGC [23] which perform graph filtering and find a consensus graph for spectral clustering, and GRACE [27] that is a general graph convolution clustering method;
- NCut [37] and Metis [55] that apply traditional graph clustering methods over the aggregation of the adjacency matrices of all graph layers in the input multiplex graph;
- ATNCut and ATMetis that apply NCut and Metis to the aggregated matrix of all layers' adjacency matrices and the attribute KNN graph; Infomap [56], Louvain [57], k-MQI [58] and k-Nibble [59] in the same way;

– CESNA [3] that treats the aggregated adjacency matrix of all layers as an attributed graph.

For all competitors, we adopt the default parameter settings as suggested in their respective papers. Hyperparameters for AMGC algorithms MCGC and MAGC are tuned as instructed in the corresponding papers, and we report the best results acquired. As for ANCKA on attributed hypergraphs, i.e., AHCKA [6], 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 $\tau$ 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 $\beta = 0.4$. In ANCKA, for attributed graphs and multiplex graphs, we fix $K = 50$, except for large datasets TWeibo and Amazon2M with $K = 10$. In particular, we find it necessary to adjust the $\beta$ parameter for certain instances following the practice in recent works [22, 23, 27]. $\beta$ is set to 0.5 for Cora and Wiki and 0.4 on Citeseer-UG, Citeseer-DG, TWeibo, and Amazon2M. We tune $\beta$ in [0.1, 0.9] by step size 0.1 for multiplex graphs. All the remaining hyperparameters in ANCKA follow the default setting of AHCKA. The parameter settings in GPU-based ANCKA-GPU are identical to ANCKA.

## 8.2 Performance evaluation

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

### 8.2.1 Quality evaluation

The clustering quality is measured by 4 classic metrics including overall accuracy (Acc), average per-class F1 score (F1), normalized mutual information (NMI), and adjusted Rand index (ARI). The former three metrics are in the range [0, 1], whereas ARI ranges from −0.5 to 1. We also sort all methods by each metric and calculate their average Quality Rank for AHC, AGC, and AMGC, provided in the last column of Tables 3, 5 and 6.

**AHC.** Tables 2 and 3 present the Acc, F1, NMI, and ARI scores of each method on small and medium/large attributed hypergraph datasets, respectively. The first observation from Tables 2 and 3 is that ANCKA on attributed hyergraphs (i.e., AHCKA) consistently achieves outstanding performance over all competitors on all datasets under almost all metrics, often by a significant margin. ANCKA has a quality rank of 1.3, much higher than the runner-up ATMetis (4.9) and GRAC (5.2). On all the four small datasets (i.e., Query, Cora-CA, Cora-CC, and Citeseer), ANCKA outperforms the best competitors (underlined in Table 2) by at least 1.9% in terms of Acc and NMI. On all the four medium/large attributed hypergraphs (i.e., 20News, DBLP, Amazon, and MAG-PM), ANCKA also yields remarkable improvements upon the competitors, with percentages up to 12.6%, 10.4%, 6.5%, 13.6% in Acc, F1, NMI, and ARI respectively. Few exceptions exist, where ANCKA 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 ANCKA over various attributed hypergraphs from different application domains, e.g., web queries, news messages, and review data. The performance of ANCKA is ascribed to our optimizations based on KNN augmentation and MHC in Sects. 3 and 4, and the framework for generating high-quality BCM matrices in Sect. 5.

**AGC.** Tables 4 and 5 present the Acc, F1, NMI, and ARI scores of each method on all attributed graphs for AGC task. ANCKA consistently outperforms existing competitors under most metrics, though few exceptions exist where ANCKA is comparable to the best. ANCKA has a quality rank of 1.3, much higher than the runner-up with quality rank 4.0. For example, on Citeseer-UG in Table 4, ANCKA achieves higher Acc, F1, NMI and ARI than the runner-up performance underlined. On the two large datasets, TWeibo and Amazon2M in Table 5, ANCKA also produces clusters with high quality, while GNMF, AGCGCN, and FGC run out of memory or cannot finish within 12 h. Notably, on Amazon2M, ANCKA surpasses all methods on all metrics except F1 (0.006 behind ATMetis) while achieving 0.494 accuracy (runner-up is Louvain at 0.463) and 0.545 ARI (runner-up is Louvain at 0.520). The effectiveness of ANCKA validates the versatility of the proposed techniques for different clustering tasks, e.g., AGC. Besides, ATMetis and ATNCut generally outperform Metis and NCut in AGC performance, respectively, exhibiting the efficacy of the proposed KNN augmentation.

**AMGC.** Table 6 reports the Acc, F1, NMI, and ARI scores of all methods on all attributed multiplex graphs. ANCKA has the best quality rank. As shown, on ACM and DBLP-MG, ANCKA achieves the best clustering quality among all methods under all metrics, with NMI and ARI leading by at least 3% on ACM, while being the second best in three metrics on IMDB. As shown later in Table 9, on these datasets, ANCKA is faster than existing native AMGC methods by at least an order of magnitude. With the intuitive design of random walk transition matrix $\mathbf{P}_N$ on multiplex graphs in Sect. 6.1, ANCKA can utilize the proposed KNN augmentation, clustering objective, and optimization techniques to maintain its excellent performance on the AMGC task.

### 8.2.2 Efficiency evaluation

Tables 7, 8 and 9 report the runtime (in seconds, with KNN construction included) and memory overhead (in Gigabytes), for AHC, AGC, and AMGC, respectively. For ease of comparing the trade-off between quality and efficiency, the last column of Tables 7, 8 and 9 contains the corresponding quality ranks from Tables 3, 5 and 6, respectively. In each table, the methods are separated into two categories: *non-native* methods extended from other clustering problems and *native* methods for the corresponding task. For instance, in Table 7, there are 4 native AHC methods in the last 4 rows, while the non-native methods are in the rows above.

In Tables 7, 8, and 9, although certain non-native methods are efficient, their quality ranks in terms of clustering quality are typically low. Hence, in the following, we mainly compare the efficiency of ANCKA against the native methods for each task. A method is terminated early if it runs out of memory (OOM) or cannot finish within 12 h.

**AHC.** In Table 7, compared with native AHC methods, we can observe that ANCKA is significantly faster on most datasets, often by orders of magnitude. For example, on a small graph Citeseer, ANCKA takes 0.635 seconds, while the fastest AHC competitor GRAC needs 13.15 seconds, meaning that ANCKA is 20.7$\times$ faster. On large attributed hypergraphs including Amazon and MAG-PM, most existing AHC solutions fail to finish due to the OOM errors, whereas ANCKA achieves 11.4$\times$ and 2.6$\times$ speedup over the only viable native AHC competitor GRAC on Amazon and MAG-PM, respectively. An exception is 20News, which contains a paucity of hyperedges (100 hyperedges), where ANCKA is slower than GRAC. Recall that in Table 3, compared to ANCKA, GRAC yields far inferior accuracy in terms of clustering on 20News, which highlights the advantages of ANCKA over GRAC. Additionally, while ATMetis is fast, it achieves an average quality rank of 4.9, which falls short of the 1.7 quality rank attained by ANCKA. As shown in Tables 2 and 3, ANCKA surpasses ATMetis in all metrics but one. Moreover, ATMetis encounters OOM on Amazon. As for the memory consumption (including the space to store hypergraphs), observe that ANCKA has comparable memory overheads with the native AHC competitors on small graphs and up to 3.1$\times$ memory reduction on medium/large graphs.

**AGC.** In Table 8 for AGC, ANCKA has comparable running time to ACMin, a recent AGC method that is optimized for efficiency, while being faster than the other native AGC methods. However, the quality rank of ANCKA is 1.3, much higher than 5.6 of ACMin. Specifically, in Tables 4 and 5, ANCKA consistently achieves better clustering quality than ACMin on all six attributed graphs under all metrics. Moreover, ANCKA remains to be the runner-up in terms of running time on the first five datasets, and is the fastest on the largest Amazon2M for clustering. Memory-wise, ANCKA consumes a moderate amount of memory that stays below 1GB over the first four small datasets and achieves decent performance on two large datasets, TWeibo and Amazon2M.

**AMGC.** In Table 9, ANCKA achieves a significant speedup ratio over the native AMGC baselines, often by an order of magnitude, while being memory efficient. Specifically, ANCKA achieves a speedup of 15.0$\times$, 13.9$\times$, and 9.5$\times$, compared to the runner-up native AMGC methods MAGC and GRACE. The memory consumption of ANCKA is also less than the majority of existing native AMGC methods.

### 8.2.3 Evaluation on ANCKA-GPU

We compare the cluster quality and efficiency of the CPU-based ANCKA against ANCKA-GPU in Sect. 7, with results reported in Table 10 for the three tasks (AHC, AGC, and AMGC) over all datasets. First, observe that ANCKA-GPU achieves similarly high-quality cluster results as the CPU-based ANCKA across all datasets for all three tasks, and the quality difference between ANCKA-GPU and ANCKA are often negligible, in terms of Acc, F1, NMI, and ARI.

The last column of Table 10 provides the running time of ANCKA-GPU and ANCKA with 16 CPU threads. For the AHC task, the speedup of ANCKA-GPU is less significant on the small attributed hypergraphs (Query, Cora-CA, Cora-CC, and Citeseer). We ascribe this to the numerous SVD operations on small $k\times k$ matrices in Discretize-GPU, as it has been known that small dimensions of input matrices may hurt the efficiency of GPU-based SVD [63]. On medium/large attributed hypergraphs (20News, DBLP, Amazon, and MAG-PM), the GPU-accelerated version, ANCKA-GPU, achieves speedup ratios of 30.5, 70.2, 8.44, and 11.4, respectively, over the CPU version ANCKA. The high speedup ratios of ANCKA-GPU, often exceeding an order of magnitude, validate the efficiency of the technical designs elaborated in Sect. 7, especially on large-scale hypergraphs. For the AGC task, similarly, on small attributed graphs, Cora, Citeseer-UG, Wiki, and Citeseer-DG, ANCKA-GPU is faster than ANCKA while the speedup ratio is usually below 10, due to the same reason explained above. On large attributed graphs (TWeibo and Amazon2M), ANCKA-GPU is more efficient than ANCKA by an order of magnitude. For the AMGC task, ANCKA-GPU is also consistently faster than ANCKA on all attributed multiplex graphs. The memory consumption of ANCKA-GPU is measured by GPU video memory (VRAM), while that of ANCKA is by RAM, and the consumption is reported in the second last column of Table 10 in GBs. The memory usage of ANCKA-GPU and ANCKA is not directly comparable, due to the different computational architectures and libraries used on GPUs and CPUs. Note that the major memory consumption of our implementations

**Table 2** Attributed hypergraph clustering (AHC) 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 |
| `HAC` | 0.541 | 0.575 | 0.453 | 0.173 | 0.374 | 0.336 | 0.234 | 0.096 | 0.374 | 0.336 | 0.234 | 0.096 | 0.376 | 0.352 | 0.188 | 0.083 |
| `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 |
| `k-MQI` | 0.222 | 0.071 | 0.019 | −0.001 | 0.304 | 0.070 | 0.005 | 0.001 | 0.302 | 0.069 | 0.005 | 0.000 | 0.212 | 0.059 | 0.003 | 0.000 |
| `k-Nibble` | 0.252 | 0.121 | 0.025 | 0.008 | 0.321 | 0.119 | 0.070 | 0.060 | 0.391 | 0.165 | 0.155 | 0.098 | 0.345 | 0.170 | 0.139 | 0.102 |
| `Infomap` | 0.235 | 0.215 | 0.017 | 0.002 | 0.514 | 0.464 | 0.343 | 0.266 | 0.541 | 0.479 | 0.393 | **0.347** | 0.491 | 0.463 | 0.263 | 0.221 |
| `Louvain` | 0.239 | 0.218 | 0.017 | 0.003 | 0.501 | 0.430 | 0.332 | 0.217 | 0.569 | **0.546** | 0.373 | 0.269 | 0.570 | 0.486 | 0.319 | 0.308 |
| `CESNA` | 0.222 | 0.191 | 0.024 | 0.002 | 0.305 | 0.092 | 0.030 | 0.000 | 0.378 | 0.240 | 0.140 | 0.053 | 0.206 | 0.060 | 0.012 | 0.000 |
| `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` | 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 |
| `ANCKA` | **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** |

The best is in bold and the runner-up is underlined

**Table 3** Attributed hypergraph clustering (AHC) quality on medium/large datasets

| Algorithm | 20News | | | | DBLP | | | | Amazon | | | | MAG-PM | | | | Quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | F1 | NMI | ARI | Acc | F1 | NMI | ARI | Acc | F1 | NMI | ARI | Acc | F1 | NMI | ARI | rank |
| 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 | 15.5 |
| 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 | 13.8 |
| 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 | 10.9 |
| 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 | 10.1 |
| HAC | 0.430 | 0.382 | 0.237 | 0.058 | 0.571 | 0.532 | 0.372 | 0.310 | OOM | | | | OOM | | | | 11.1 |
| 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 | 13.8 |
| 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 | 10.8 |
| 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 | 4.9 |
| 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 | 5.7 |
| k-MQI | 0.336 | 0.126 | 0.000 | 0.000 | 0.271 | 0.071 | 0.000 | 0.000 | OOM | | | | 0.252 | 0.018 | 0.000 | 0.000 | 17.1 |
| k-Nibble | 0.338 | 0.129 | 0.002 | 0.000 | 0.254 | 0.086 | 0.028 | 0.006 | OOM | | | | 0.252 | 0.019 | 0.000 | 0.000 | 14.3 |
| Infomap | 0.338 | 0.129 | 0.004 | 0.000 | 0.595 | 0.573 | 0.488 | 0.404 | OOM | | | | 0.398 | 0.172 | 0.380 | 0.248 | 9.5 |
| Louvain | 0.633 | 0.522 | 0.304 | 0.323 | 0.643 | 0.580 | 0.554 | 0.470 | OOM | | | | OOM | | | | 8.3 |
| CESNA | 0.379 | 0.350 | 0.086 | 0.047 | 0.272 | 0.072 | 0.001 | 0.000 | >12 h | | | | >12 h | | | | 15.5 |
| 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 | 6.3 |
| 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** | 6.8 |
| GNMF | 0.436 | 0.271 | 0.070 | 0.061 | 0.613 | 0.506 | 0.417 | 0.407 | OOM | | | | OOM | | | | 10.6 |
| JNMF | 0.582 | 0.423 | 0.247 | 0.241 | 0.618 | 0.588 | 0.447 | 0.396 | OOM | | | | OOM | | | | 11.0 |
| 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 | 5.3 |
| ANCKA | **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 | **1.3** |

The best is in bold and the runner-up is underlined

**Table 4** Attributed graph clustering (AGC) quality on cora, Citeseer-UG & Wiki

| Algorithm | Cora | | | | Citeseer-UG | | | | Wiki | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | F1 | NMI | ARI | Acc | F1 | NMI | ARI | Acc | F1 | NMI | ARI |
| `Metis` | 0.448 | 0.436 | 0.330 | 0.238 | 0.391 | 0.380 | 0.155 | 0.131 | 0.408 | 0.364 | 0.351 | 0.206 |
| `NCut` | 0.298 | 0.072 | 0.012 | −0.003 | 0.218 | 0.087 | 0.009 | 0.004 | 0.172 | 0.025 | 0.016 | 0.000 |
| `k-means` | 0.318 | 0.295 | 0.151 | 0.072 | 0.454 | 0.429 | 0.223 | 0.173 | 0.275 | 0.176 | 0.272 | 0.081 |
| `HAC` | 0.372 | 0.328 | 0.219 | 0.095 | 0.422 | 0.383 | 0.190 | 0.139 | 0.449 | 0.375 | 0.437 | 0.185 |
| `ATMetis` | 0.471 | 0.448 | 0.317 | 0.241 | 0.586 | 0.566 | 0.337 | 0.318 | 0.506 | 0.440 | 0.505 | 0.336 |
| `ATNCut` | 0.417 | 0.403 | 0.271 | 0.112 | 0.409 | 0.374 | 0.212 | 0.090 | 0.424 | 0.381 | 0.471 | 0.150 |
| `k-MQI` | 0.302 | 0.068 | 0.004 | 0.000 | 0.211 | 0.059 | 0.003 | 0.000 | 0.169 | 0.021 | 0.013 | 0.000 |
| `k-Nibble` | 0.378 | 0.167 | 0.138 | 0.041 | 0.281 | 0.151 | 0.097 | 0.018 | 0.217 | 0.105 | 0.114 | 0.021 |
| `Infomap` | 0.569 | 0.503 | 0.455 | 0.301 | 0.590 | 0.546 | 0.312 | 0.317 | 0.467 | 0.417 | 0.468 | 0.290 |
| `Louvain` | 0.671 | 0.640 | 0.474 | 0.397 | 0.680 | 0.621 | 0.426 | 0.413 | **0.611** | **0.513** | **0.572** | **0.427** |
| `CESNA` | 0.320 | 0.251 | 0.198 | 0.053 | 0.212 | 0.074 | 0.022 | 0.001 | 0.450 | 0.332 | 0.371 | 0.251 |
| `GNMF` | 0.554 | 0.450 | 0.413 | 0.283 | 0.562 | 0.478 | 0.296 | 0.301 | 0.486 | 0.353 | 0.504 | 0.352 |
| `AGCGCN` | 0.689 | 0.655 | 0.531 | 0.446 | 0.675 | 0.630 | 0.418 | 0.424 | 0.446 | 0.384 | 0.422 | 0.108 |
| `FGC` | 0.693 | 0.590 | 0.541 | 0.470 | 0.682 | 0.635 | 0.431 | 0.439 | 0.513 | 0.420 | 0.484 | 0.239 |
| `ACMin` | 0.655 | 0.558 | 0.492 | 0.417 | 0.674 | 0.636 | 0.416 | 0.429 | 0.450 | 0.281 | 0.391 | 0.255 |
| `GRACE` | 0.720 | **0.723** | 0.533 | 0.456 | 0.678 | 0.634 | 0.416 | 0.431 | 0.603 | 0.453 | 0.526 | 0.302 |
| `ANCKA` | **0.723** | 0.686 | **0.556** | **0.484** | **0.691** | **0.651** | **0.438** | **0.450** | 0.551 | 0.467 | 0.543 | 0.353 |

The best is in bold and the runner-up is underlined

is in the KNN augmentation step. On small or medium-sized datasets, e.g., Query and Cora-CA, VRAM usage by `ANCKA-GPU` is higher than the RAM usage by `ANCKA`. The reason is that `ANCKA-GPU` uses GPU-based Faiss for nearest-neighbor search and Faiss allocates about 700MB of VRAM for temporary storage. On large datasets, `ANCKA` requires a substantial RAM space due to the implementation of the ScaNN algorithm for KNN, while GPU-based Faiss in `ANCKA-GPU` requires less VRAM space.

Then we enhance `GRACE` [27] with GPU acceleration using CuPy and cuML libraries, resulting in `GRACE-GPU` for comparison. We also compare with the GPU-based implementation of the Spectral Modularity Maximization [64] clustering method dubbed as `SMM-GPU`, which operates on the graph adjacency matrix for AGC (or clique expansion of the hypergraph for AHC, or the sum of multiplex adjacency matrices for AMGC) with the attribute KNN augmentation. The results for AHC, AGC, and AMGC are presented in Tables 11, 12 and 13, respectively. On the first six smaller datasets in Table 11 for AHC, `SMM-GPU` exhibits lower quality in terms of Acc, F1, NMI, and ARI, despite comparable efficiency to `ANCKA-GPU`, which delivers significantly better clustering quality. `ANCKA-GPU` outperforms `GRACE-GPU` in both quality and efficiency across all AHC datasets. Notably, on large datasets Amazon and MAG-PM in Table 11, `ANCKA-GPU` efficiently produces satisfactory clusters, whereas `GRACE-GPU` and `SMM-GPU` encounter out-of-memory due to their requirement to expand hypergraphs into graphs. Similar observations are made for AGC and AMGC in Tables 12 and 13. Similar patterns are observed for AGC and AMGC in Tables 12 and 13. In these tasks, `ANCKA-GPU` delivers superior clustering quality and efficiency on most datasets, except IMDB where `ANCKA-GPU` is the second best, while `SMM-GPU` yields lower-quality outcomes and `GRACE-GPU` falls behind our method in speed. We conclude that `ANCKA-GPU` offers high clustering quality with remarkable efficiency.

## 8.3 Experimental analysis

**Varying** $K$. Figure 7 depicts the Acc, F1, NMI scores, and the KNN computation time of `ANCKA` on 8 attributed hypergraphs (AHC) when varying $K$ from 2 to 1000. We can make the following observations. First, on most hypergraphs, the clustering accuracies of `ANCKA` 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 `ANCKA` 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 Fig. 2. Moreover, as $K$ goes up, the time of KNN construction increases on all datasets. Figures 8 and 9 show the Acc, F1, NMI scores and KNN computation time of `ANCKA` on the 6 attributed graphs and 3 attributed multiplex graphs for AGC and AMGC, respectively, when varying $K$ from 2 to 1000. On small graphs in Fig. 8a–d and

**Table 5** Attributed graph clustering (AGC) quality on Citeseer-DG, Tweibo & Amazon2M

| Algorithm | Citeseer-DG | | | | Tweibo | | | | Amazon2M | | | | **Quality** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | F1 | NMI | ARI | Acc | F1 | NMI | ARI | Acc | F1 | NMI | ARI | rank |
| Metis | 0.410 | 0.397 | 0.175 | 0.155 | 0.141 | 0.093 | 0.007 | 0.004 | 0.223 | 0.163 | 0.277 | 0.080 | 10.0 |
| NCut | 0.278 | 0.069 | 0.009 | −0.004 | 0.427 | 0.067 | 0.000 | 0.000 | 0.136 | 0.016 | 0.004 | −0.005 | 13.8 |
| k-means | 0.440 | 0.419 | 0.209 | 0.158 | 0.277 | 0.108 | 0.013 | −0.011 | 0.178 | 0.055 | 0.100 | 0.008 | 10.9 |
| HAC | 0.461 | 0.444 | 0.208 | 0.153 | OOM | | | | OOM | | | | 11.7 |
| ATMetis | 0.594 | 0.575 | 0.366 | 0.340 | 0.131 | 0.086 | 0.005 | 0.003 | 0.267 | **0.197** | 0.411 | 0.127 | 6.8 |
| ATNCut | 0.465 | 0.378 | 0.277 | 0.120 | 0.420 | 0.078 | 0.003 | 0.008 | 0.272 | 0.010 | 0.003 | −0.001 | 10.2 |
| k-MQI | 0.212 | 0.059 | 0.003 | 0.000 | 0.411 | 0.048 | 0.001 | 0.000 | 0.273 | 0.009 | 0.000 | 0.000 | 14.3 |
| k-Nibble | 0.283 | 0.151 | 0.098 | 0.019 | 0.428 | 0.067 | 0.000 | 0.000 | 0.375 | 0.042 | 0.015 | 0.004 | 12.0 |
| Infomap | 0.621 | 0.565 | 0.357 | 0.368 | 0.417 | 0.084 | 0.000 | 0.001 | 0.357 | 0.191 | 0.424 | 0.214 | 6.6 |
| Louvain | 0.682 | 0.617 | 0.419 | 0.408 | 0.271 | 0.113 | 0.015 | 0.007 | 0.463 | 0.154 | 0.429 | 0.520 | 4.0 |
| CESNA | 0.213 | 0.074 | 0.022 | 0.001 | >12 h | | | | 0.273 | 0.009 | 0.000 | 0.000 | 12.8 |
| GNMF | 0.570 | 0.526 | 0.347 | 0.353 | OOM | | | | OOM | | | | 9.6 |
| AGCGCN | 0.672 | 0.624 | 0.416 | 0.420 | OOM | | | | OOM | | | | 8.3 |
| FGC | 0.684 | 0.635 | 0.436 | 0.444 | >12 h | | | | >12 h | | | | 6.5 |
| ACMin | 0.677 | 0.633 | 0.420 | 0.433 | 0.399 | 0.109 | 0.004 | 0.012 | 0.318 | 0.182 | 0.342 | 0.126 | 5.6 |
| GRACE | 0.684 | 0.638 | 0.424 | 0.440 | 0.292 | 0.119 | **0.026** | −0.009 | 0.271 | 0.154 | 0.340 | 0.118 | 4.0 |
| ANCKA | **0.696** | **0.651** | **0.444** | **0.460** | **0.433** | **0.129** | 0.023 | **0.019** | **0.494** | 0.191 | **0.441** | **0.545** | **1.3** |

The best is in bold and the runner-up is underlined

9, the cluster quality increases from 2 to 50, and then declines on datasets such as Citeseer-UG, Wiki, and ACM. On large datasets TWeibo and Amazon2M in Fig. 8e and f, a turning point appears around $K = 10$. Therefore, we set $K$ to be 50 and 10 on these small and large datasets, respectively.

**Varying $\boldsymbol{\beta}$**. Recall that in the generalized $(\alpha, \beta, \gamma)$-random walk model, the parameter $\beta$ is used to balance the combination of topological proximities from graph topology $\mathcal{N}_O$ and the attribute similarities from KNN graph $\mathcal{G}_K$. Figure 10 displays the AHC performance of ANCKA on 8 attributed hypergraph datasets when $\beta$ varies from 0 to 1. When $\beta = 0$, ANCKA degrades to a hypergraph clustering method without the consideration of any attribute information, whereas ANCKA only clusters the KNN graph $\mathcal{G}_K$ regardless of the topology structure in $\mathcal{H}$ if $\beta = 1$. From Fig. 10, we can see a large $\beta$ (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 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, ANCKA achieves the best clustering quality with small $\beta$ in [0.1, 0.4], indicating graph topology has higher weights on Amazon and MAG-PM. Figures 11 and 12 report the Acc, F1, and NMI scores on AGC and AMGC tasks respectively. Similarly, when $\beta$ increases from 0, the cluster quality generally improves, then becomes stable around 0.4 and 0.5, and decreases when $\beta$ is large and close to 1. On DBLP-MG in Fig. 12, the highest clustering quality can be acquired with a small $\beta$ around 0.1. We infer that node attributes in this dataset are of limited significance for clustering, while on ACM and IMDB, the best quality is achieved when $\beta$ appropriately balances graph topology and attributes.

**Varying $\boldsymbol{\gamma}$**. We evaluate ANCKA in terms of AHC quality and running time when varying $\gamma$. Figure 13 displays the Acc, F1, NMI, and time on two representative datasets when $\gamma$ varies from 1 to 5. The results on other datasets are similar and thus are omitted for space. Observe that in practice the Acc, F1, and NMI scores obtained by ANCKA first increase and then remain stable when $\gamma$ is beyond 3 and 2 on Cora-CC and Citeseer, respectively. By contrast, the running time goes up as $\gamma$ increases. Therefore, we set $\gamma = 3$ in experiments.

**Effectiveness Evaluation of InitBCM and Discretize.** On attributed hypergraphs, to verify the effectiveness of InitBCM for the BCM initialization, we compare ANCKA with the ablated version ANCKA-random-init, where the BCM matrix $\mathbf{Y}^{(0)}$ is initialized at random. In Table 14, ANCKA obtains remarkable improvements over ANCKA-random-init in Acc, F1, and NMI in comparable processing time. For instance, on Amazon, ANCKA outperforms ANCKA-random-init by a large margin of 3.7% Acc, 19.5% F1, and 6.8% NMI with 24 s less to process. On MAG-PM, ANCKA needs additional time compared to ANCKA-random-init. The reason is that ANCKA-random-init starts with a low-quality BCM and converges to local optimum solutions with subop-

**Table 6** Attributed multiplex graph clustering (AMGC) quality

| Algorithm | ACM | | | | IMDB | | | | DBLP-MG | | | | Quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | F1 | NMI | ARI | Acc | F1 | NMI | ARI | Acc | F1 | NMI | ARI | rank |
| Metis | 0.648 | 0.651 | 0.389 | 0.369 | 0.376 | 0.374 | 0.004 | 0.004 | 0.864 | 0.860 | 0.660 | 0.688 | 11.6 |
| NCut | 0.350 | 0.174 | 0.003 | 0.000 | 0.378 | 0.185 | 0.002 | 0.000 | 0.299 | 0.125 | 0.012 | −0.001 | 15.3 |
| k-means | 0.679 | 0.681 | 0.320 | 0.312 | 0.525 | 0.531 | 0.146 | 0.139 | 0.368 | 0.285 | 0.083 | 0.060 | 10.1 |
| HAC | 0.576 | 0.557 | 0.234 | 0.222 | 0.483 | 0.462 | 0.100 | 0.101 | 0.381 | 0.304 | 0.131 | 0.070 | 11.4 |
| ATMetis | 0.755 | 0.757 | 0.510 | 0.490 | 0.546 | <u>0.551</u> | 0.161 | 0.152 | 0.868 | 0.864 | 0.669 | 0.697 | 6.5 |
| ATNCut | 0.778 | 0.775 | 0.462 | 0.465 | 0.499 | 0.466 | 0.154 | 0.165 | 0.360 | 0.285 | 0.104 | 0.023 | 8.8 |
| k-MQI | 0.351 | 0.174 | 0.001 | 0.000 | 0.377 | 0.183 | 0.001 | 0.000 | 0.295 | 0.115 | 0.002 | 0.000 | 15.8 |
| k-Nibble | 0.343 | 0.221 | 0.018 | 0.001 | 0.370 | 0.251 | 0.022 | 0.005 | 0.295 | 0.115 | 0.002 | 0.000 | 15.2 |
| Infomap | 0.653 | 0.665 | 0.418 | 0.353 | 0.412 | 0.362 | 0.027 | 0.025 | 0.296 | 0.116 | 0.002 | 0.000 | 12.6 |
| Louvain | 0.659 | 0.670 | 0.422 | 0.364 | 0.452 | 0.392 | 0.057 | 0.065 | 0.909 | 0.900 | 0.731 | 0.788 | 8.1 |
| CESNA | 0.624 | 0.593 | 0.405 | 0.330 | 0.377 | 0.329 | 0.006 | 0.007 | 0.827 | 0.820 | 0.583 | 0.603 | 12.0 |
| O2MAC | 0.895 | 0.897 | 0.667 | 0.716 | 0.547 | 0.550 | 0.135 | 0.139 | 0.873 | 0.865 | 0.669 | 0.705 | 5.5 |
| HDMI | 0.900 | 0.899 | 0.695 | 0.732 | 0.541 | 0.547 | 0.162 | 0.142 | 0.895 | 0.885 | 0.706 | 0.761 | 4.7 |
| MCGC | <u>0.915</u> | <u>0.916</u> | <u>0.709</u> | <u>0.763</u> | 0.567 | 0.545 | 0.164 | 0.186 | 0.902 | 0.895 | 0.716 | 0.771 | 3.5 |
| MAGC | 0.872 | 0.872 | 0.597 | 0.659 | 0.484 | 0.424 | 0.057 | 0.062 | <u>0.928</u> | <u>0.923</u> | <u>0.771</u> | <u>0.827</u> | 6.0 |
| GRACE | 0.889 | 0.891 | 0.651 | 0.698 | **0.629** | **0.629** | **0.185** | **0.205** | 0.923 | 0.918 | 0.767 | 0.817 | <u>3.0</u> |
| ANCKA | **0.928** | **0.928** | **0.739** | **0.796** | <u>0.576</u> | 0.544 | <u>0.176</u> | <u>0.195</u> | **0.933** | **0.929** | **0.785** | **0.839** | **1.7** |

The best is in bold and the runner-up is underlined

timal MHC, whereas ANCKA 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 ANCKA to transform $k$ leading eigenvectors $\mathbf{Q}$ to BCM matrix $\mathbf{Y}$. Table 14 reports the accuracy of ANCKA and a variant ANCKA-k-means obtained by replacing Discretize in ANCKA with k-means on all datasets. It can be observed that compared with ANCKA-k-means, ANCKA is able to output high-quality BCM matrices $\mathbf{Y}$ with substantially higher clustering accuracy scores while being up to 3.2× faster. The ablation results on AGC and AMGC are in Tables 15 and 16, respectively. Regarding clustering quality (Acc, F1, NMI), Table 15 shows that for AGC, ANCKA surpasses its ablated counterparts on all datasets across most effectiveness metrics, except for the Citeseer datasets. For example, ANCKA with InitBCM achieves an Acc that is 4.2% higher than ANCKA-random-init on Amazon2M. In Table 16 for AMGC, ANCKA performs the best on all the three datasets. For efficiency in Tables 15 and 16, ANCKA is similar to ANCKA-random-init, while ANCKA-k-means is slower. These results confirm the effectiveness of the proposed techniques for AGC and AMGC.

## 8.4 Convergence analysis

We provide an empirical analysis pertinent to the convergence of ANCKA for attributed hypergraph clustering. 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 $\phi_t$) of the BCM matrix $\mathbf{Y}^{(t)}$ generated in each $t$-th iteration of ANCKA and ANCKA-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 ANCKA. Figure 14 shows the MHC $\phi_t$, Acc, F1, and NMI scores based on the BCM matrix of each iteration in ANCKA, as well as the MHC of ANCKA-random-init over all datasets. Notably, MHC $\phi_t$ experiences a sharp decline when $t$ increases from 0 to 50 on most hypergraphs, while the Acc, F1, and NMI results have significant growth. Moreover, compared to MHC with random init, MHC curves of ANCKA are mostly lower (better) on all datasets under the same $t$-th iteration. These phenomena demonstrate the effectiveness of InitBCM in facilitating fast convergence of ANCKA. 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 indicates that adding more iterations does not necessarily ensure better solutions. Hence, the early termination proposed in ANCKA can serve as an effective approach to remedy this issue.

## 8.5 Runtime analysis

Figure 15 reports time breakdown of ANCKA and ANCKA-GPU into four parts: KNN construction, orthogonal iterations, dis-

**Table 7** Efficiency of attributed hypergraph clustering (AHC) (time in seconds, RAM in GBs)

| Algorithm | Query | | Cora-CA | | Cora-CC | | Citeseer | | 20News | | DBLP | | Amazon | | MAG-PM | | Quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | RAM | Time | RAM | Time | RAM | Time | RAM | Time | RAM | Time | RAM | Time | RAM | Time | RAM | rank |
| HNCut | 0.057 | 0.139 | 0.582 | 0.141 | 0.654 | 0.158 | 0.528 | 0.172 | 0.116 | 0.134 | 4.282 | 0.625 | 477.1 | 3.291 | 666.8 | 4.256 | 13.8 |
| KaHyPar | 1.556 | 0.105 | 0.375 | 0.132 | 0.285 | 0.129 | 0.292 | 0.159 | 1.516 | 0.119 | 3.795 | 0.606 | 3707 | 37.14 | 556.9 | 9.487 | 10.9 |
| ATHNCut | 0.393 | 0.178 | 0.650 | 0.428 | 0.657 | 0.429 | 0.835 | 1.031 | 4.935 | 2.164 | 35.68 | 3.589 | 685.9 | 54.24 | 789.3 | 57.36 | 10.8 |
| ATMetis | 0.081 | 0.125 | 0.238 | 0.282 | 0.239 | 0.283 | 0.389 | 0.307 | 13.55 | 2.966 | 26.85 | 3.308 | OOM | | 557.9 | 64.20 | 4.9 |
| ATKaHyPar | 1.668 | 0.128 | 1.610 | 0.225 | 1.543 | 0.225 | 1.733 | 0.304 | 11.47 | 2.400 | 50.32 | 3.256 | 5529 | 54.28 | 1509 | 57.41 | 5.7 |
| k-MQI | 0.104 | 0.243 | 0.361 | 0.352 | 0.376 | 0.363 | 0.418 | 0.427 | 11.23 | 3.866 | 28.98 | 3.397 | OOM | | 1567 | 60.54 | 17.1 |
| k-Nibble | 0.151 | 0.224 | 5.827 | 0.655 | 5.888 | 0.667 | 21.83 | 0.885 | 44.55 | 8.975 | 1338 | 51.77 | OOM | | 3858 | 281.6 | 14.3 |
| Infomap | 0.221 | 0.191 | 0.742 | 0.291 | 0.611 | 0.293 | 0.719 | 0.363 | 556.1 | 21.43 | 43.50 | 3.263 | OOM | | 11,756 | 200.9 | 9.5 |
| Louvain | 0.732 | 0.195 | 1.915 | 0.232 | 0.735 | 0.242 | 1.911 | 0.313 | 1567 | 21.77 | 70.15 | 3.313 | OOM | | OOM | | 8.3 |
| CESNA | 0.620 | 0.119 | 2.400 | 0.134 | 9.816 | 0.137 | 3.251 | 0.164 | 7643 | 0.157 | 92.04 | 0.617 | >12 h | | >12 h | | 15.5 |
| GNMF | 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 | | 10.6 |
| 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 | | 11.0 |
| 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 | 14,662 | 175.1 | 3504 | 92.69 | 5.3 |
| ANCKA | **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** | **1.3** |

The Quality Rank column is from Table 3. Among all native AHC methods in the last 4 rows,
The best is in bold, and the runner-up is underlined

**Table 8** Efficiency of attributed graph clustering (AGC) algorithms (time in seconds, RAM in GBs)

| Algorithm | Cora | | Citeseer-UG | | Wiki | | Citeseer-DG | | Tweibo | | Amazon2M | | Quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | RAM | Time | RAM | Time | RAM | Time | RAM | Time | RAM | Time | RAM | rank |
| Metis | 0.006 | 0.203 | 0.009 | 0.347 | 0.242 | 0.489 | 0.006 | 0.316 | 121.6 | 4.688 | 46.40 | 8.150 | 10.0 |
| NCut | 0.072 | 0.198 | 0.350 | 0.326 | 0.087 | 0.475 | 0.321 | 0.347 | 409.3 | 4.662 | 874.8 | 8.132 | 13.8 |
| ATMetis | 0.688 | 0.352 | 0.571 | 0.533 | 0.807 | 0.591 | 0.398 | 0.275 | 360.0 | 13.58 | 130.4 | 16.63 | 6.8 |
| ATNCut | 0.469 | 0.351 | 0.589 | 0.501 | 0.902 | 0.560 | 0.548 | 0.258 | 334.6 | 13.66 | 502.5 | 16.58 | 10.2 |
| k-MQI | 0.382 | 0.438 | 0.488 | 0.585 | 0.721 | 0.502 | 0.348 | 0.447 | 2442 | 29.90 | 1453 | 19.65 | 14.3 |
| k-Nibble | 4.656 | 0.495 | 19.21 | 0.626 | 13.86 | 0.550 | 18.99 | 0.685 | 3826 | 102.3 | 3587 | 89.14 | 12.0 |
| Infomap | 1.265 | 0.398 | 1.668 | 0.544 | 1.499 | 0.739 | 1.556 | 0.310 | 6701 | 97.48 | 4155 | 45.75 | 6.6 |
| Louvain | 6.773 | 0.371 | 7.319 | 0.503 | 4.527 | 0.669 | 6.584 | 0.572 | 10,010 | 84.50 | 21,696 | 72.80 | 4.0 |
| CESNA | 12.81 | **0.144** | 35.21 | **0.167** | 354.9 | **0.162** | 28.12 | **0.178** | >12 h | | 1931 | **7.701** | 12.8 |
| GNMF | 13.18 | 0.269 | 37.01 | 0.397 | 22.38 | 0.579 | 42.81 | 0.438 | OOM | | OOM | | 9.6 |
| AGCGCN | 5.842 | 0.960 | 33.34 | 2.120 | 5.965 | 1.003 | 34.18 | 2.326 | OOM | | OOM | | 8.3 |
| FGC | 29.68 | 1.998 | 225.7 | 3.273 | 50.93 | 3.080 | 44.93 | 3.571 | >12 h | | >12 h | | 6.5 |
| ACMin | **0.368** | 0.164 | **0.400** | 0.177 | 3.646 | 0.380 | **0.556** | 0.234 | **1098** | **18.61** | 5300 | 20.21 | 5.6 |
| GRACE | 5.589 | 0.651 | 21.82 | 1.793 | 16.78 | 1.740 | 15.23 | 1.960 | 2317 | 60.44 | 3162 | 39.71 | 4.0 |
| ANCKA | 1.251 | 0.369 | 1.587 | 0.517 | **0.907** | 0.706 | 0.838 | 0.280 | 1318 | 19.89 | **1708** | 17.01 | **1.3** |

The Quality Rank column is from Table 5. Among all native AGC methods in the last 7 rows,
The best is in bold, and the runner-up is underlined

**Table 9** Efficiency of attributed multiplex graph clustering (AMGC) algorithms (time in seconds, RAM in GBs)

| Algorithm | ACM | | IMDB | | DBLP-MG | | Quality |
|---|---|---|---|---|---|---|---|
| | Time | RAM | Time | RAM | Time | RAM | rank |
| Metis | 0.477 | 0.382 | 0.037 | 0.375 | 1.798 | 0.602 | 11.6 |
| NCut | 0.761 | 0.392 | 0.123 | 0.384 | 2.218 | 0.611 | 15.3 |
| ATMetis | 1.418 | 1.034 | 1.181 | 1.134 | 2.441 | 0.672 | 6.5 |
| ATNCut | 1.324 | 1.037 | 1.236 | 1.141 | 2.587 | 0.675 | 8.8 |
| k-MQI | 1.033 | 1.143 | 1.064 | 1.319 | 1.048 | 0.957 | 15.8 |
| k-Nibble | 7.230 | 0.696 | 10.32 | 0.766 | 3.999 | 1.109 | 15.2 |
| Infomap | 17.78 | 1.547 | 3.624 | 1.260 | 48.45 | 3.883 | 12.6 |
| Louvain | 43.91 | 1.300 | 9.537 | 1.151 | 158.0 | 3.948 | 8.1 |
| CESNA | 68.85 | 0.309 | 32.28 | 0.372 | 819.2 | 0.534 | 12.0 |
| O2MAC | 115.0 | 1.691 | 679.1 | 2.109 | 684.1 | 2.638 | 5.5 |
| HDMI | 161.2 | 2.902 | 245.9 | 2.980 | 537.8 | 3.162 | 4.7 |
| MCGC | 748.2 | 1.697 | 1552 | 2.414 | 2245 | 3.283 | 3.5 |
| MAGC | 26.10 | 1.301 | 33.69 | 1.908 | 35.98 | 2.665 | 6.0 |
| GRACE | 110.1 | 1.173 | 21.81 | **1.341** | 49.33 | **0.672** | 3.0 |
| ANCKA | **1.738** | **1.062** | **1.574** | 1.485 | **3.766** | 0.691 | **1.7** |

The Quality Rank column is from Table 6. Among all native AMGC methods in the last 6 rows,
The best is in bold, and the runner-up is underlined

cretization, and greedy initialization and MHC evaluation on all attributed hypergraphs. We first explain the results of ANCKA on CPUs. On all datasets, the four parts in ANCKA all take considerable time to process, except 20News and DBLP, where KNN construction dominates, since 20News and DBLP contain many nodes but relatively few edges. Then, we compare the time breakdown of ANCKA-GPU with ANCKA. On small attributed hypergraphs (Query, Cora-CA, Cora-CC, and Citeseer) in Figs. 15a, b, c, and d, observe that ANCKA-GPU significantly reduces the time for KNN, while the other time costs are on par with that of ANCKA, which is consistent with the results in Sect. 8.2.3. On medium-sized/large attributed hypergraphs in Fig. 15e, f, g, and h, ANCKA-GPU significantly improves the efficiency on all of KNN construction, orthogonal iterations, discretization, greedy initialization and MHC evaluation. From the results on Amazon and MAG-PM, we observe that the scalability of ANCKA-GPU is primarily constrained by KNN construction,

**Table 10** Evaluation between ANCKA and ANCKA-GPU

| Task | Dataset | Acc | | F1 | | NMI | | ARI | | Mem | | Time | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CPU | GPU | CPU | GPU | CPU | GPU | CPU | GPU | CPU | GPU | CPU | GPU (Speedup) |
| AHC | Query | 0.715 | 0.719 | 0.662 | 0.664 | 0.645 | 0.666 | 0.571 | 0.578 | 0.161 | 1.083 | 0.342 | 0.230 (1.49×) |
| | Cora-CA | 0.651 | 0.653 | 0.608 | 0.610 | 0.462 | 0.469 | 0.406 | 0.411 | 0.231 | 1.096 | 0.402 | 0.265 (1.52×) |
| | Cora-CC | 0.592 | 0.580 | 0.520 | 0.535 | 0.412 | 0.395 | 0.338 | 0.311 | 0.232 | 1.098 | 0.416 | 0.296 (1.41×) |
| | Citeseer | 0.662 | 0.668 | 0.615 | 0.620 | 0.392 | 0.387 | 0.397 | 0.410 | 0.317 | 1.128 | 0.635 | 0.575 (1.10×) |
| | 20News | 0.712 | 0.712 | 0.658 | 0.666 | 0.409 | 0.407 | 0.469 | 0.465 | 0.383 | 1.094 | 8.176 | 0.268 (30.5×) |
| | DBLP | 0.797 | 0.808 | 0.774 | 0.787 | 0.632 | 0.643 | 0.632 | 0.646 | 0.998 | 1.321 | 41.50 | 0.591 (70.2×) |
| | Amazon | 0.660 | 0.648 | 0.492 | 0.487 | 0.630 | 0.636 | 0.524 | 0.509 | 56.71 | 11.16 | 1286 | 152.3 (8.44×) |
| | MAG-PM | 0.566 | 0.559 | 0.405 | 0.393 | 0.561 | 0.545 | 0.471 | 0.454 | 59.25 | 11.35 | 1371 | 120.2 (11.4×) |
| AGC | Cora | 0.723 | 0.683 | 0.686 | 0.621 | 0.556 | 0.533 | 0.484 | 0.470 | 0.369 | 1.120 | 1.251 | 0.213 (5.87×) |
| | Citeseer-UG | 0.691 | 0.690 | 0.651 | 0.649 | 0.438 | 0.437 | 0.450 | 0.451 | 0.517 | 1.153 | 1.587 | 0.507 (3.13×) |
| | Wiki | 0.551 | 0.560 | 0.467 | 0.487 | 0.543 | 0.547 | 0.353 | 0.368 | 0.706 | 1.151 | 0.907 | 0.357 (2.57×) |
| | Citeseer-DG | 0.696 | 0.694 | 0.651 | 0.652 | 0.444 | 0.441 | 0.460 | 0.454 | 0.280 | 1.159 | 0.838 | 0.508 (1.65×) |
| | TWeibo | 0.433 | 0.434 | 0.129 | 0.126 | 0.023 | 0.022 | 0.019 | 0.016 | 19.89 | 16.73 | 1318 | 105.0 (12.6×) |
| | Amazon2M | 0.494 | 0.496 | 0.191 | 0.194 | 0.441 | 0.437 | 0.545 | 0.544 | 17.01 | 18.08 | 1708 | 158.9 (10.8×) |
| AMGC | ACM | 0.928 | 0.924 | 0.928 | 0.924 | 0.739 | 0.730 | 0.796 | 0.786 | 1.062 | 1.267 | 1.738 | 0.190 (9.15×) |
| | IMDB | 0.576 | 0.553 | 0.544 | 0.510 | 0.176 | 0.166 | 0.195 | 0.184 | 1.485 | 1.136 | 1.574 | 0.236 (6.67×) |
| | DBLP-MG | 0.933 | 0.935 | 0.929 | 0.931 | 0.785 | 0.791 | 0.839 | 0.842 | 0.691 | 1.787 | 3.766 | 0.587 (6.42×) |

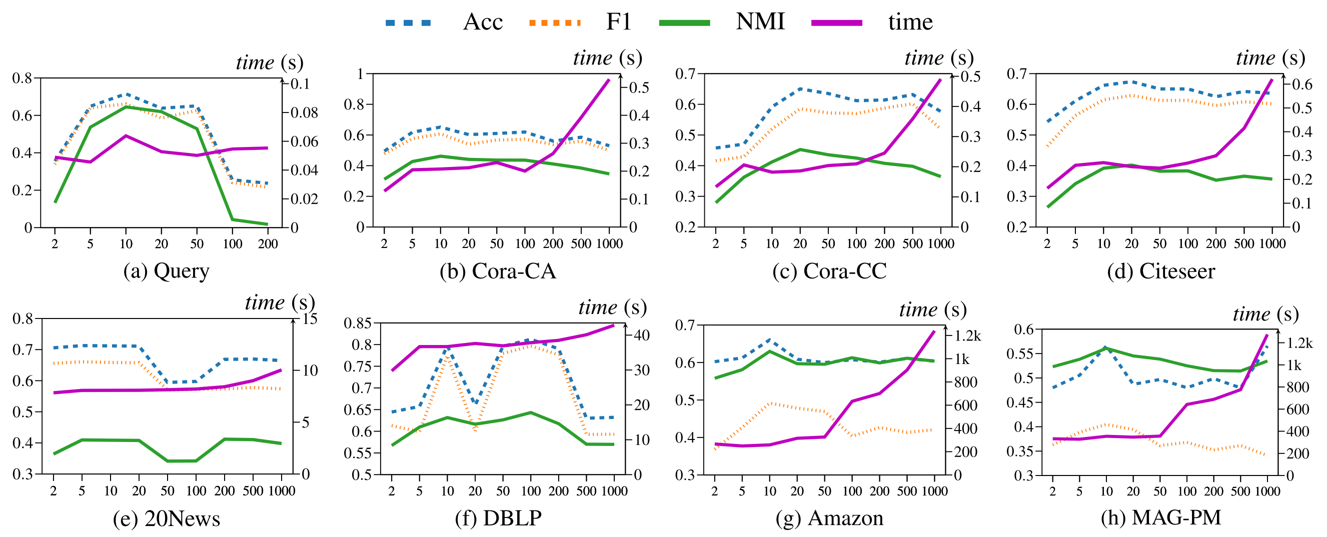

**Fig. 7** Varying $K$ for AHC (best viewed in color)

while the overhead of the CPU-based ANCKA is more evenly distributed across the four parts.

## 9 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 [65] and KaHyPar [14]. 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) [52] is a conductance measure for hypergraph clusters from which the normalized hypergraph Laplacian $\Delta = \mathbf{I} - \Theta$ is derived for spectral

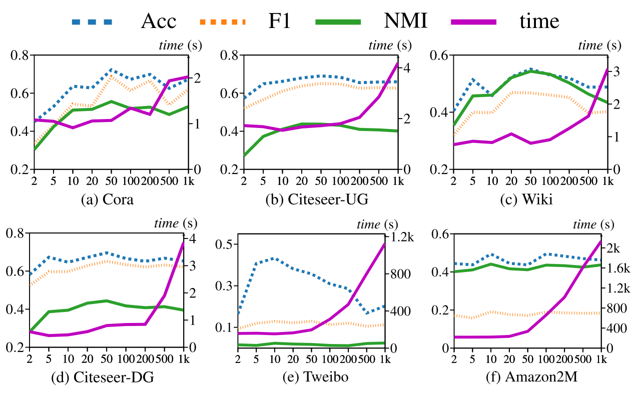

**Fig. 8** Varying $\beta$ for AHC (best viewed in color)

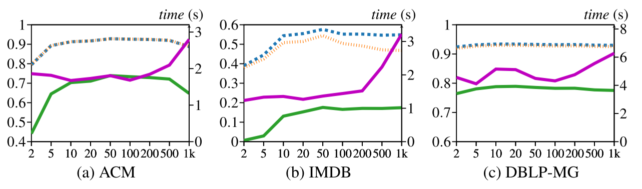

**Fig. 9** Varying $K$ for AGC (best viewed in color)

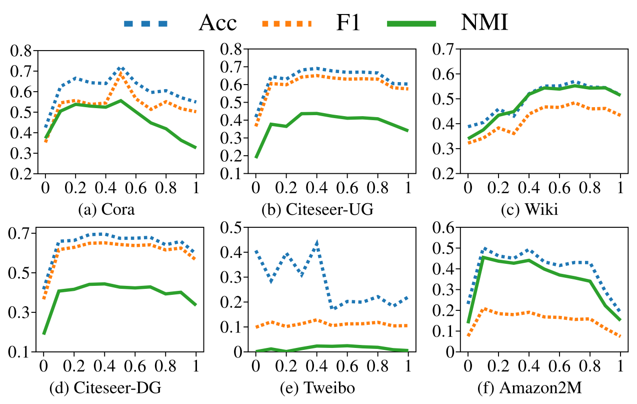

**Fig. 11** Varying $\beta$ for AGC (best viewed in color)

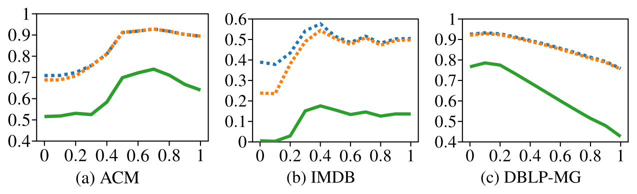

**Fig. 12** Varying $\beta$ for AMGC (best viewed in color)

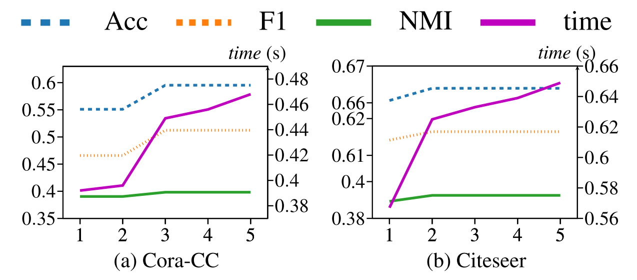

**Fig. 13** Varying $\gamma$ on Attributed Hypergraphs

clustering, where $\Theta = \mathbf{D}_V^{-1/2}\mathbf{H}^T\mathbf{D}_E^{-1}\mathbf{H}\mathbf{D}_V^{-1/2}$. Alternatively, `hGraclus` [5] optimizes the HNCut objective using a multi-level kernel K-means algorithm. Non-negative matrix factorization has also been applied to hypergraph clustering [15]. 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 [66] to find the cluster based on hypergraph Personalized PageRank (PPR) values. In this paper, we focus on global clustering, a different problem from local clustering.

**Attributed Hypergraph Clustering.** There exist studies designing dedicated clustering algorithms on attributed hypergraphs. `JNMF` [1] is an AHC algorithm based on non-negative matrix factorization (NMF). With normalized hypergraph Laplacian [52] matrix $\Delta = \mathbf{I} - \Theta$ and attribute matrix $\mathbf{X}$, `JNMF` optimizes the following joint objective that includes a basic NMF part as well as a symmetric NMF part: $\min_{\mathbf{W},\mathbf{M},\tilde{\mathbf{M}}\geq 0} ||\mathbf{X} - \mathbf{W}\mathbf{M}||_F^2 + \alpha||\Theta - \tilde{\mathbf{M}}^{\mathrm{T}}\mathbf{M}||_F^2 + \beta||\tilde{\mathbf{M}} - \mathbf{M}||_F^2$. With optimization using block coordinate descent (BCD) scheme, the matrix $\mathbf{M}$ is expected to encode cluster memberships. `MEGA` [5] 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` [60] algorithm is originally proposed for high dimensional data clustering, while the authors of [11] extend its objective with the hypergraph normalized Laplacian [52] so that it spawns baseline methods for AHC. Although

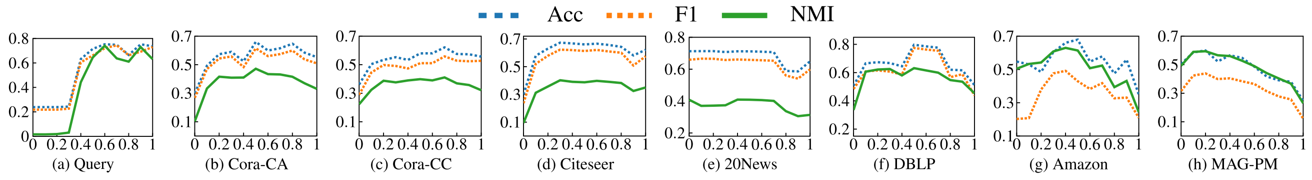

**Fig. 10** Varying $K$ for AMGC (best viewed in color)

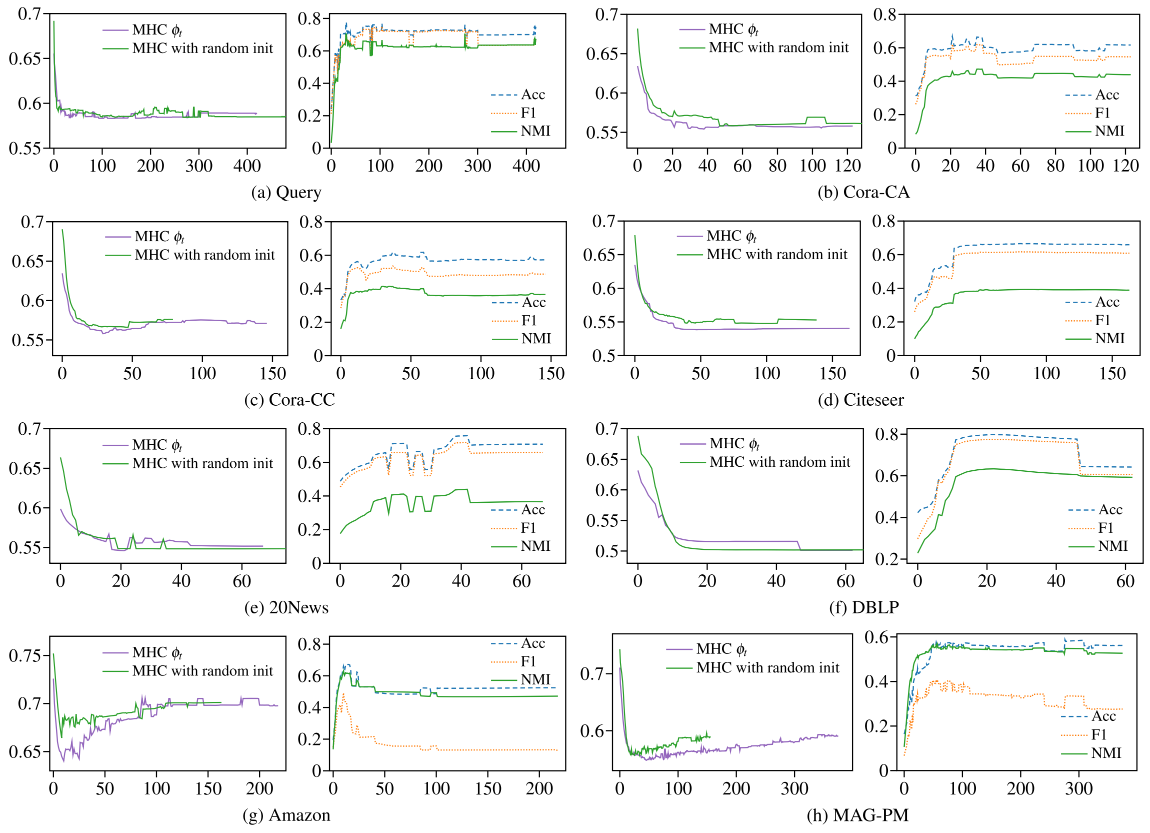

**Fig. 14** Convergence Analysis (best viewed in color)

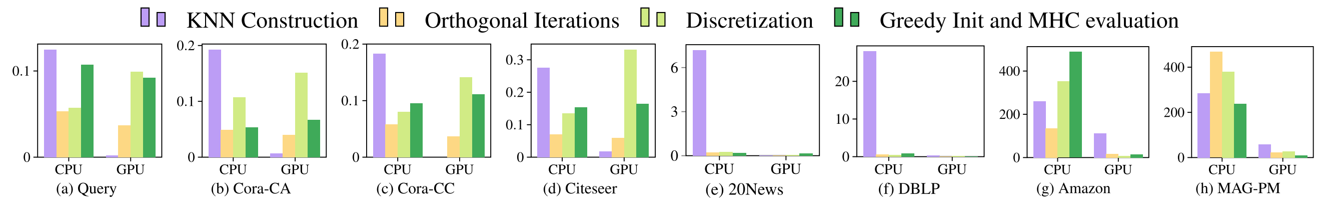

**Fig. 15** Runtime breakdown of CPU-based ANCKA and ANCKA-GPU in seconds

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 [11] performs hypergraph convolution [46] on node attributes, which resembles the hypergraph diffusion process with mediators [67]. Then clusters are predicted from the propagated features via a spectral algorithm.

**Attributed Graph Clustering.** There exists a collection of studies on attributed graph clustering. Some studies perform attributed graph clustering by adopting probabilistic models to combine graph structure with attributes, including discriminative models such as PCL-DC [68] and generative models such as BAGC [2]. 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. GNMF [60] is an NMF-based algorithm that enhances performance by modifying the Laplacian regularizer used in traditional NMF to utilize the Laplacian matrix constructed from the graph structure. Within the random walk framework, SA-Cluster [12] 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 [13] 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 [17], AGCGCN [50] 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

**Table 11** Additional GPU baselines for AHC

| Algorithm | Query | | | | | | Cora-CA | | | | | | Cora-CC | | | | | | Citeseer | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | F1 | NMI | ARI | Mem | Time | Acc | F1 | NMI | ARI | Mem | Time | Acc | F1 | NMI | ARI | Mem | Time | Acc | F1 | NMI | ARI | Mem | Time |
| SMM-GPU | 0.237 | 0.191 | 0.012 | 0.004 | **1.073** | 0.381 | 0.155 | 0.039 | 0.022 | 0.012 | **1.089** | **0.220** | 0.155 | 0.039 | 0.085 | 0.016 | **1.088** | **0.251** | 0.212 | 0.059 | 0.055 | 0.002 | 1.329 | **0.265** |
| GRACE-GPU | 0.420 | 0.435 | 0.204 | 0.076 | 1.823 | 1.326 | 0.589 | 0.583 | 0.368 | 0.296 | 1.956 | 4.870 | 0.550 | 0.503 | 0.346 | 0.253 | 1.904 | 2.466 | 0.512 | 0.465 | 0.280 | 0.271 | 2.064 | 5.356 |
| ANCKA-GPU | **0.719** | **0.664** | **0.666** | **0.578** | 1.083 | **0.230** | **0.653** | **0.610** | **0.469** | **0.411** | 1.096 | 0.265 | **0.580** | **0.535** | **0.395** | **0.311** | 1.098 | 0.296 | **0.668** | **0.620** | **0.387** | **0.410** | **1.128** | 0.575 |
| Algorithm | 20News | | | | | | DBLP | | | | | | Amazon | | | | | | MAGPM | | | | | |
| | Acc | F1 | NMI | ARI | Mem | Time | Acc | F1 | NMI | ARI | Mem | Time | Acc | F1 | NMI | ARI | Mem | Time | Acc | F1 | NMI | ARI | Mem | Time |
| SMM-GPU | 0.440 | 0.430 | 0.192 | 0.155 | 3.120 | 1.816 | 0.160 | 0.046 | 0.013 | 0.000 | 2.543 | 0.703 | OOM | | | | | | OOM | | | | | |
| GRACE-GPU | 0.361 | 0.316 | 0.079 | 0.022 | 1.920 | 2.408 | 0.681 | 0.695 | 0.543 | 0.443 | 3.170 | 47.90 | OOM | | | | | | OOM | | | | | |
| ANCKA-GPU | **0.712** | **0.666** | **0.407** | **0.465** | **1.094** | **0.268** | **0.808** | **0.787** | **0.643** | **0.646** | **1.321** | **0.591** | **0.648** | **0.487** | **0.636** | **0.510** | **11.16** | **152.3** | **0.559** | **0.393** | **0.545** | **0.454** | **11.35** | **120.2** |

The best is in bold and the runner-up is underlined

**Table 12** Additional GPU baselines for AGC

| Algorithm | Cora | | | | | | Citeseer-UG | | | | | | Wiki | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | F1 | NMI | ARI | Mem | Time | Acc | F1 | NMI | ARI | Mem | Time | Acc | F1 | NMI | ARI | Mem | Time |
| SMM-GPU | 0.408 | 0.325 | 0.227 | 0.161 | 2.585 | 0.293 | 0.437 | 0.373 | 0.223 | 0.204 | 2.752 | **0.405** | 0.533 | 0.433 | 0.503 | 0.345 | 2.773 | 0.360 |
| GRACE-GPU | **0.698** | **0.694** | 0.498 | 0.429 | 1.973 | 3.235 | 0.681 | 0.636 | 0.421 | 0.435 | 2.189 | 6.614 | 0.527 | 0.329 | 0.500 | 0.286 | 1.976 | 8.494 |
| ANCKA-GPU | 0.683 | 0.621 | **0.533** | **0.470** | **1.120** | **0.213** | **0.690** | **0.649** | **0.437** | **0.451** | **1.153** | 0.503 | **0.560** | **0.487** | **0.547** | **0.368** | **1.151** | **0.357** |
| Algorithm | Citeseer-DG | | | | | | Tweibo | | | | | | Amazon2M | | | | | |
| | Acc | F1 | NMI | ARI | Mem | Time | Acc | F1 | NMI | ARI | Mem | Time | Acc | F1 | NMI | ARI | Mem | Time |
| SMM-GPU | 0.438 | 0.375 | 0.226 | 0.206 | 2.774 | **0.382** | 0.389 | 0.098 | 0.012 | −0.013 | 23.58 | **32.88** | 0.206 | 0.052 | 0.092 | 0.023 | **9.287** | **62.55** |
| GRACE-GPU | 0.685 | 0.636 | 0.427 | 0.441 | 2.162 | 3.745 | OOM | | | | | | 0.282 | 0.171 | 0.352 | 0.120 | 22.08 | 529.3 |
| ANCKA-GPU | **0.694** | **0.652** | **0.441** | **0.454** | **1.159** | 0.508 | **0.434** | **0.126** | **0.022** | **0.016** | **16.73** | 105.0 | **0.496** | **0.194** | **0.437** | **0.544** | 18.08 | 158.9 |

The best is in bold and the runner-up is underlined

**Table 13** Additional GPU baselines for AMGC

| Algorithm | ACM | | | | | | IMDB | | | | | | DBLP-MG | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | F1 | NMI | ARI | Mem | Time | Acc | F1 | NMI | ARI | Mem | Time | Acc | F1 | NMI | ARI | Mem | Time |
| `SMM-GPU` | 0.615 | 0.580 | 0.337 | 0.331 | 2.903 | 0.293 | 0.544 | 0.440 | **0.194** | **0.197** | 2.797 | 0.354 | 0.557 | 0.356 | 0.427 | 0.383 | 3.251 | 0.633 |
| `GRACE-GPU` | 0.888 | 0.890 | 0.648 | 0.694 | 2.157 | 7.802 | 0.532 | **0.532** | 0.115 | 0.112 | 2.352 | 12.37 | 0.922 | 0.917 | 0.765 | 0.815 | 2.641 | 3.569 |
| `ANCKA-GPU` | **0.924** | **0.924** | **0.730** | **0.786** | **1.267** | **0.190** | **0.553** | 0.510 | 0.166 | 0.184 | **1.136** | **0.236** | **0.935** | **0.931** | **0.791** | **0.842** | **1.787** | **0.587** |

The best is in bold and the runner-up is underlined

**Table 14** Ablation Analysis on AHC (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 |
| `ANCKA`-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 |
| `ANCKA-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 |
| `ANCKA` | **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 |
| `ANCKA`-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** |
| `ANCKA-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 |
| `ANCKA` | **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 |

The best is in bold and the runner-up is underlined

**Table 15** Ablation analysis on AGC (time in seconds)

| Algorithm | Cora | | | | Citeseer-UG | | | | Wiki | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | F1 | NMI | Time | Acc | F1 | NMI | Time | Acc | F1 | NMI | Time |
| ANCKA-random-init | 0.676 | 0.619 | 0.544 | **0.869** | 0.681 | **0.681** | 0.435 | **1.436** | 0.507 | 0.436 | 0.529 | 1.082 |
| ANCKA-k-means | 0.597 | 0.456 | 0.511 | 1.254 | 0.684 | 0.631 | **0.440** | 1.915 | 0.459 | 0.385 | 0.506 | 3.030 |
| ANCKA | **0.723** | **0.686** | **0.556** | 1.251 | **0.691** | 0.651 | 0.438 | 1.587 | **0.551** | **0.467** | **0.543** | **0.907** |
| Algorithm | Citeseer-DG | | | | Tweibo | | | | Amazon2M | | | |
| | Acc | F1 | NMI | Time | Acc | F1 | NMI | Time | Acc | F1 | NMI | Time |
| ANCKA-random-init | 0.689 | 0.649 | 0.443 | **0.685** | 0.364 | 0.094 | 0.005 | **1068** | 0.452 | 0.188 | 0.405 | **1269** |
| ANCKA-k-means | 0.689 | 0.636 | **0.445** | 2.005 | 0.428 | 0.067 | 0.000 | 1837 | 0.429 | 0.107 | 0.406 | 3420 |
| ANCKA | **0.696** | **0.651** | 0.444 | 0.838 | **0.433** | **0.129** | **0.023** | 1318 | **0.494** | **0.191** | **0.441** | 1708 |

The best is in bold and the runner-up is underlined

**Table 16** Ablation Analysis on AMGC (Time in Seconds)

| Algorithm | ACM | | | | IMDB | | | | DBLP-MG | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | F1 | NMI | Time | Acc | F1 | NMI | Time | Acc | F1 | NMI | Time |
| ANCKA-random-init | 0.923 | 0.924 | 0.728 | **1.546** | 0.536 | 0.482 | 0.165 | **1.131** | 0.932 | 0.928 | 0.783 | 4.391 |
| ANCKA-k-means | 0.926 | 0.927 | 0.738 | 1.818 | 0.383 | 0.203 | 0.005 | 1.703 | 0.926 | 0.920 | 0.774 | 4.719 |
| ANCKA | **0.928** | **0.928** | **0.739** | 1.738 | **0.576** | **0.544** | **0.176** | 1.574 | **0.933** | **0.929** | **0.785** | **3.766** |

The best is in bold and the runner-up is underlined

also design heuristics to prevent propagated features from over-smoothing that undermines cluster quality. GRACE [27] adopts graph convolution on node attributes to fuse all available information and perform a spectral algorithm based on GRAC [11]. FGC [62] exploits both node features and structure information via graph convolution and applies spectral clustering on a fine-grained graph that encodes higher-order relations.

**Attributed Multiplex Graph Clustering.** Via unsupervised learning on attributed multiplex graphs, neural network models can learn node embeddings for clustering, e.g., O2MAC [25] and HDMI [24]. GRACE [27] constructs a multiplex graph Laplacian and uses this matrix for graph convolution. Other methods find a single graph that encodes the node proximity relations in all graph layers and attributes. MCGC [22] performs graph filtering on attributes and learns a consensus graph leveraging contrastive regularization, while MAGC [23] exploits higher-order proximity to learn consensus graphs without deep neural networks.

## 10 Conclusion

This paper presents ANCKA, a versatile, effective, and efficient attributed network clustering method for AHC, AGC, and AMGC computation. The improvements of ANCKA over existing solutions in terms of efficiency and effectiveness is attributed to: (i) an effective KNN augmentation strategy to exploit useful attribute information, (ii) a novel problem formulation based on a random walk model, and (iii) an efficient iterative optimization framework with speedup techniques. To further boost the efficiency, we leverage GPUs and develop ANCKA-GPU that is faster than its CPU-parallel counterpart ANCKA on large datasets, while retaining high cluster quality. We conduct extensive experiments over real-world data to validate the outstanding performance of our methods. In the future, we plan to extend ANCKA to cope with evolving attributed networks and enhance its scalability via distributed KNN construction and matrix computation.

**Acknowledgements** The work described in this paper was fully supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (No. PolyU 25201221, PolyU 15205224, PolyU 15200023). Jieming Shi is supported by NSFC No. 62202404. Renchi Yang is supported by the NSFC YSF grant (No. 62302414) and Hong Kong RGC ECS grant (No. 22202623). Jun Luo is supported by The Innovation and Technology Fund (Ref. ITP/067/23LP). This work is supported by Tencent Technology Co., Ltd. P0048511.

**Funding** Open access funding provided by The Hong Kong Polytechnic University

## References

1. Du, R., Drake, B., Park, H.: Hybrid clustering based on content and connection structure using joint nonnegative matrix factorization. J. Glob. Optim. **74**(4), 861–877 (2019)
2. Xu, Z., Ke, Y., Wang, Y., Cheng, H., Cheng, J.: A model-based approach to attributed graph clustering. In: SIGMOD, pp. 505–516 (2012)
3. Yang, J., McAuley, J., Leskovec, J.: Community detection in networks with node attributes. In: ICDM, pp. 1151–1156 (2013)
4. Shi, J., Mamoulis, N., Wu, D., Cheung, DW.: Density-based place clustering in geo-social networks. In: SIGMOD Conference, ACM, pp. 99–110 (2014)
5. Whang, J.J., Du, R., Jung, S., Lee, G., Drake, B., Liu, Q., Kang, S., Park, H.: MEGA: multi-view semi-supervised clustering of hypergraphs. VLDB **13**(5), 698–711 (2020)
6. Li, Y., Yang, R., Shi, J.: Efficient and effective attributed hypergraph clustering via k-nearest neighbor augmentation. PACMMOD **1**, 1–23 (2023)
7. Gaudelet, T., Malod-Dognin, N., Przulj, N.: Higher-order molecular organization as a source of biological function. Bioinformatics **34**(17), i944–i953 (2018)
8. Wu, L., Hu, Y., Zhou, Y., Wang, H., Luo, X., Wang, Z., Zhang, F., Ren, K.: Towards understanding and demystifying bitcoin mixing services. In: WWW, pp. 33–44 (2021)
9. Huang, L., Wang, C.D., Yu, P.S.: Higher order connection enhanced community detection in adversarial multiview networks. IEEE Trans. Cybern. **53**(5), 3060–3074 (2021)
10. Wu, M.J., Gao, Y.L., Liu, J.X., Zheng, C.H., Wang, J.: Integrative hypergraph regularization principal component analysis for sample clustering and co-expression genes network analysis on multi-omics data. IEEE JBHI **24**(6), 1823–1834 (2020)
11. Fanseu Kamhoua, B., Zhang, L., Ma, K., Cheng, J., Li, B., Han, B.: HyperGraph convolution based attributed hypergraph clustering. In: CIKM, pp. 453–463 (2021)
12. Zhou, Y., Cheng, H., Yu, JX.: Clustering large attributed graphs: an efficient incremental approach. In: ICDM, pp. 689–698 (2010)
13. Yang, R., Shi, J., Yang, Y., Huang, K., Zhang, S., Xiao, X.: Effective and scalable clustering on massive attributed graphs. In: WWW, pp. 3675–3687 (2021)
14. Schlag, S., Heuer, T., Gottesbüren, L., Akhremtsev, Y., Schulz, C., Sanders, P.: High-quality hypergraph partitioning. J. Exp. Algorithmics (2022). https://doi.org/10.1145/3529090
15. Hayashi, K., Aksoy, SG., Park, CH., Park, H.: Hypergraph random walks, Laplacians, and clustering. In: CIKM, pp. 495–504 (2020)
16. Kumar, T., Vaidyanathan, S., Ananthapadmanabhan, H., Parthasarathy, S., Ravindran, B.: Hypergraph clustering: a modularity maximization approach. arXiv:1812.10869 (2018)
17. Kipf, TN., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017)
18. Peng, L., Wang, X., Zhu, X.: Unsupervised multiplex graph learning with complementary and consistent information. In: ACM MM, pp. 454–462 (2023)

19. Pensky, M., Wang, Y.: Clustering of diverse multiplex networks. arXiv:2110.05308 (2021)
20. Fortunato, S.: Community detection in graphs. arXiv:0906.0612 (2009)
21. Guimerà, R., Amaral, L.A.N.: Functional cartography of complex metabolic networks. Nature **433**, 895–900 (2005)
22. Pan, E., Kang, Z.: (2021). Multi-view contrastive graph clustering. In: NeurIPS
23. Lin, Z., Kang, Z., Zhang, L., Tian, L.: Multi-view attributed graph clustering. TKDE **35**, 1872–1880 (2021)
24. Jing, B., Park, C., Tong, H.: Hdmi: high-order deep multiplex infomax. In: WWW, pp. 2414–2424 (2021)
25. Fan, S., Wang, X., Shi, C., Lu, E., Lin, K., Wang, B.: One2multi graph autoencoder for multi-view graph clustering. In: WWW, pp. 3070–3076 (2020)
26. DeFord, DR., Pauls, SD.: Spectral clustering methods for multiplex networks. arXiv:1703.05355 (2017)
27. Kamhoua, B.F., Zhang, L., Ma, K., Cheng, J., Li, B., Han, B.: Grace: a general graph convolution framework for attributed graph clustering. ACM TKDD **17**, 1–31 (2022)
28. Cheng, H., Zhou, Y., Yu, J.X.: Clustering large attributed graphs: a balance between structural and attribute similarities. ACM TKDD **5**(2), 1–33 (2011)
29. Guo, R., Sun, P., Lindgren, E., Geng, Q., Simcha, D., Chern, F., Kumar, S.: Accelerating large-scale inference with anisotropic vector quantization. In: ICML (2020)
30. Johnson, J., Douze, M., Jégou, H.: Billion-scale similarity search with GPUs. IEEE TBD **7**(3), 535–547 (2019)
31. Tong, H., Faloutsos, C., Pan, Jy.: Fast random walk with restart and its applications. In: ICDM, pp. 613–622 (2006)
32. Jung, J., Park, N., Lee, S., Kang, U.: Bepi: fast and memory-efficient method for billion-scale random walk with restart. In: SIGMOD, pp. 789–804 (2017)
33. Shi, J., Yang, R., Jin, T., Xiao, X., Yang, Y.: Realtime top-k personalized pagerank over large graphs on gpus. VLDB **13**(1), 15–28 (2019)
34. Park, H., Jung, J., Kang, U.: A comparative study of matrix factorization and random walk with restart in recommender systems. In: IEEE BigData, pp. 756–765 (2017)
35. Allen Zhu, Z., Lattanzi, S., Mirrokni, V.: A local algorithm for finding well-connected clusters. In: ICML (2013)
36. Jeh, G., Widom, J.: Scaling personalized web search. In: WWW, pp. 271–279 (2003)
37. Shi, J., Malik, J.: Normalized cuts and image segmentation. TPAMI **22**(8), 888–905 (2000)
38. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–137 (1982)
39. Won, J.H., Zhou, H., Lange, K.: Orthogonal trace-sum maximization: applications, local algorithms, and global optimality. SIAM J. Matrix Anal. Appl. **42**(2), 859–882 (2021)
40. (2024) Technical report. https://sites.google.com/view/ancka-technical-report/
41. Yu, SX., Shi, J.: Multiclass spectral clustering. In: ICCV, p. 313 (2003)
42. Saad, Y.: Numerical Methods for Large Eigenvalue Problems, revised SIAM, New Delhi (2011)
43. Rattigan, MJ., Maier, M., Jensen, D.: Graph clustering with network structure indices. In: ICML, pp. 783–790 (2007)
44. Yuster, R., Zwick, U.: Fast sparse matrix multiplication. ACM TALG **1**(1), 2–13 (2005)
45. Cook, S.: CUDA programming: a developer's guide to parallel computing with GPUs (2012)
46. Yadati, N., Nimishakavi, M., Yadav, P., Nitin, V., Louis, A., Talukdar, P.: HyperGCN: a new method of training graph convolutional networks on hypergraphs. NeurIPS **135**, 1511–1522 (2019)
47. Hein, M., Setzer, S., Jost, L., Rangapuram, SS.: The total variation on hypergraphs—learning on hypergraphs revisited. In: NeurIPS, vol. 26 (2013)
48. Ni, J., Li, J., McAuley, J.: Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In: EMNLP-IJCNLP, pp. 188–197 (2019)
49. Sinha, A., Shen, Z., Song, Y., Ma, H., Eide, D., Hsu, BP., Wang, K.: An overview of Microsoft academic service (MAS) and applications. In: WWW, pp. 243–246 (2015)
50. Zhang, X., Liu, H., Li, Q., Wu, XM.: Attributed graph clustering via adaptive graph convolution. In: IJCAI (2019)
51. Chiang, WL., Liu, X., Si, S., Li, Y., Bengio, S., Hsieh, CJ.: Cluster-gcn: an efficient algorithm for training deep and large graph convolutional networks. In: KDD (2019)
52. Zhou, D., Huang, J., Schölkopf, B.: Learning with Hypergraphs: clustering, classification, and embedding. In: NeurIPS, vol. 19 (2007)
53. Rodri, J.: On the Laplacian eigenvalues and metric parameters of hypergraphs. Linear Multilinear Algebra **50**(1), 1–14 (2002)
54. Gottesbüren, L., Heuer, T., Sanders, P.: Parallel flow-based hypergraph partitioning. In: SEA, vol. 233 (2022)
55. Karypis, G., Kumar, V.: Multilevel k-way partitioning scheme for irregular graphs. J. Parallel Distrib. Comput. **48**(1), 96–129 (1998)
56. Rosvall, M., Axelsson, D., Bergstrom, C.T.: The map equation. Eur. Phys. J. Spec. Top. **178**(1), 13–23 (2009)
57. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech. Theory Exp. **2008**(10), P10008 (2008)
58. Lang, K., Rao, S.: A flow-based method for improving the expansion or conductance of graph cuts. In: International Conference on Integer Programming and Combinatorial Optimization, pp. 325–337 (2004)
59. Andersen, R., Chung, F., Lang, K.: Local graph partitioning using Pagerank vectors. In: FOCS, pp. 475–486 (2006)
60. Cai, D., He, X., Han, J., Huang, T.S.: Graph regularized nonnegative matrix factorization for data representation. TPAMI **33**(8), 1548–1560 (2010)
61. Ward, J.H., Jr.: Hierarchical grouping to optimize an objective function. J. Am. Stat. Assoc. **58**(301), 236–244 (1963)
62. Kang, Z., Liu, Z., Pan, S., Tian, L.: Fine-grained attributed graph clustering. In: SDM (2022)
63. An, J., Wang, D.: Efficient one-sided Jacobi SVD computation on AMD GPU using OpenCL. In: ICSP (2016)
64. Newman, M.E.J.: Spectral methods for community detection and graph partitioning. Phys. Rev. E (2013). https://doi.org/10.1103/PhysRevE.88.042822
65. Karypis, G., Aggarwal, R., Kumar, V., Shekhar, S.: Multilevel hypergraph partitioning: applications in VLSI domain. IEEE TVLSI **7**(1), 69–79 (1999)
66. Takai, Y., Miyauchi, A., Ikeda, M., Yoshida, Y.: Hypergraph clustering based on Pagerank. In: KDD (2020)
67. Chan, THH., Liang, Z.: Generalizing the hypergraph Laplacian via a diffusion process with mediators. arXiv:1804.11128 (2018)
68. Yang, T., Jin, R., Chi, Y., Zhu, S.: Combining link and content for community detection: a discriminative approach. In: KDD, pp. 927–936 (2009)