On Graph Representation for Attributed Hypergraph Clustering (with Appendix)

Abstract

Attributed Hypergraph Clustering (AHC) aims at partitioning a hypergraph into clusters such that nodes in the same cluster are close to each other with both high connectedness and homogeneous attributes. Existing AHC methods are all based on matrix factorization which may incur a substantial computation cost; more importantly, they inherently require a prior knowledge of the number of clusters as an input which, if inaccurately estimated, shall lead to a significant deterioration in the clustering quality. In this paper, we propose Attributed Hypergraph Representation for Clustering (AHRC), a cluster-number-free hypergraph clustering consisting of an effective integration of the hypergraph topology and node attributes for hypergraph representation, a multi-hop modularity function for optimization, and a hypergraph sparsification for scalable computation. AHRC achieves cutting-edge clustering quality and efficiency: compared to the state-of-the-art (SOTA) AHC method on 10 real hypergraphs, AHRC obtains an average of 20% higher F-measure, 24% higher ARI, 26% higher Jaccard Similarity, 10% higher Purity, and runs 5.5× faster. As a byproduct, the intermediate result of graph representation dramatically boosts the clustering quality of SOTA contrastive-learning-based hypergraph clustering methods, showing the generality of our graph representation.

CCS Concepts

• **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Keywords

Hypergraph, Clustering, Representation, Sparsification, Modularity, Contrastive Learning

ACM Reference Format:

. 2018. On Graph Representation for Attributed Hypergraph Clustering (with Appendix). In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 17 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Attributed Graph Clustering (AGC) [21] partitions an attributed graph into a collection of disjoint node sets where each node set is called a cluster. In addition to the topological requirement imposed by traditional graph clustering, i.e., nodes in one cluster should be more closely connected to each other than to the nodes in the other clusters, AGC also expects nodes in the same cluster to have similar attributes [7]. Traditional graph clustering can be formulated as optimizations with objective functions such as normalized cut [57], conductance [34], and modularity [5], or addressed by firstly embedding the graph nodes into a vector space using eigenvalue decomposition [64] or graph neural networks [4, 63, 71] (GNNs), and then applying K-Means for clustering. AGC can be reduced to traditional graph clustering by edge re-weighting [50] based on attribute similarity, or by treating each attribute as a node in an augmented graph [80]. Both, as commented in [70], ignore the similarities between nodes that are not directly connected. Alternatively, AGC can be addressed by computing similarities between all pairs of nodes [17], integrating both topological and attributed similarity; such integration can also be achieved in a random walk model [70]. The state-of-the-art quality of AGC is achieved [39, 43] by graph contrastive learning [72], e.g., TriCL [43], which learns unsupervised representations of the graph nodes based on both graph topology and node attributes.

With graph applications engaging more with high-order connections [3], e.g., groups in social networks or author teams in citation networks [66], recent years have witnessed growing research on hypergraphs [3]. Unlike traditional graphs where each edge connects two nodes (thus called dyadic graphs), hypergraphs allow each edge to connect an arbitrary number of nodes (called hyperedges). This paper studies *Attributed Hypergraph Clustering* (AHC), aiming to partition a hypergraph into clusters such that nodes in the same cluster are close to each other with both high connectedness and homogeneous attributes. AHC has wide applications in social community detection [46], metabolic reactions analysis [38], image segmentation [36], and biological analysis [67], especially in scenarios where data involves not only high-order topological connections but also diverse node attributes. For example, in a coauthor network, a node represents an author. The node attribute forms an author profile, which could be the collection of keywords used by the author's research work, a blurb describing the research area, or a high-dimensional vector learned by a deep learning system based on the author's publications. A hyperedge represents a team of co-authors who published a paper in a joint effort. AHC identifies groups of authors with high research relevance by jointly considering their profiles and co-authorship in hyperedges. In a protein complex network, a node denotes a protein, node attributes describe the protein, and a hyperedge represents a group of proteins that form a multi-protein complex. AHC uncovers groups of proteins that share high functional similarities, providing insights into biological processes. However, existing AHC methods face two challenges in achieving both efficiency and effectiveness: the difficulty of hypergraph representation and limited scalability.

Firstly, existing AHC methods [10, 19, 32, 46] are all matrix factorization based. They require prior knowledge of the number of

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

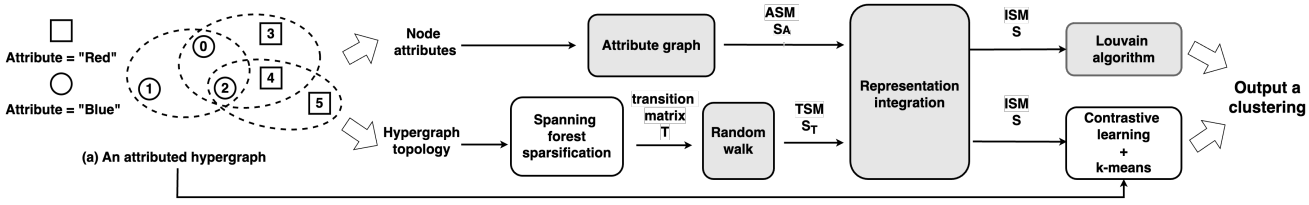


Figure 1: An overview of our pipeline AHRC

clusters to produce quality clustering and may incur substantial computation costs. Specifically, the cluster number of a desirable AHC is usually dataset-dependent and unknown in advance, which, if inaccurately estimated, shall lead to a significant deterioration in the clustering quality [65]. Besides, matrix factorization facilitated with Non-negative Matrix Factorization (NMF), Singular Value Decomposition (SVD), or eigendecomposition, leads to substantial computational and memory costs, thereby limiting scalability [32]. The state-of-the-art AHC method AHCKA [46] adopts a greedy iterative method to approximate the eigendecomposition, achieving outstanding performance; however, its algorithm design and performance are still highly sensitive to the cluster number, and the complexity of approximate eigendecomposition may hinder a further improvement on the scalability of AHCKA.

Secondly, a more effective integration of the hypergraph topology and node attributes is desirable for quality and scalable clustering. Existing works [22, 59, 78] focusing on only one of the two lack a clear pathway for integration. The integration that involves matrix operations such as NMF and SVD incurs high computation costs [10, 19, 32] and is thus not scalable. AHCKA [46] performs a multi-hop random walk where each step has a fixed probability to walk along the attribute graph – the graph where each node v is connected to the nodes whose attribute sets have the largest cosine similarity to that of v – instead of the hypergraph. However, it is unclear why the attribute similarity should be propagated through multi-hop random walks especially when the attribute graph has already considered attribute similarities among all node pairs.

Given the above challenges, this paper considers three questions. Q1) How to integrate the topological and attributed information more effectively to enhance clustering quality? Q2) How to conduct clustering without prior knowledge of cluster numbers while achieving high scalability over large attributed hypergraphs? Q3) Could one generate a graph representation for AHC that is generally applicable, e.g., can improve the clustering quality of existing learning-based methods? We provide positive answers to the above questions with Attributed Hypergraph Representation for Clustering (AHRC), a hypergraph clustering pipeline that achieves cutting-edge efficiency and effectiveness. AHRC comprehensively outperforms the state-of-the-art attributed hypergraph clustering method AHCKA [46]: averaged over 10 real hypergraphs, AHRC obtained 13% higher F-measure, 16% higher ARI, 17% higher Jaccard Similarity, 11% higher Purity, and is 5.5× faster in running time.

Figure 1 overviews the pipeline of our AHRC. Given an attributed hypergraph, AHRC computes an Attribute Similarity Matrix (ASM) S_A and a Topology Similarity Matrix (TSM) S_T to capture the node-wise relationships in terms of attributes and hypergraph topology, respectively, and integrate them into an Integrated Similarity Matrix (ISM) S. S_A is derived from the attribute graph while S_T is obtained by firstly sparsifying the hypergraph and then performing a random walk to capture multi-hop relations. AHRC formulates AHC as an optimization on the objective function of multi-hop modularity and then engages the cluster-number-free Louvain for clustering; the intermediated ISM S can alternatively be fed into other clustering methods, e.g., contrastive learning-based clustering method, for general useage. In the design of the pipeline, we find that excluding the attribute similarity from the random walk and our unique presentation integration of the attribute and hypergraph topology are highly effective in enhancing the clustering quality. To make the method scalable, we introduce a sparsification module, which dramatically improves the efficiency without deteriorating the clustering quality. Our contributions are summarized below.

(1) We propose a cluster-number-free AHC method AHRC that represents an attributed hypergraph for clustering by effectively integrating both hypergraph topology and node attributes. The graph representation allows a formulation of a multi-hop modularity as the objective function for optimization and can be of independent and general use in other clustering frameworks.
(2) AHRC adopts spanning forest sparsification to further scale up the pipeline while preserving essential features for clustering.
(3) Extensive experiments justify the outperformance of AHRC over the state-of-the-art (SOTA) AHC methods in both scalability and effectiveness. AHRC is efficient: averaged over all datasets, our AHRC speeds up the SOTA method AHCKA [46] by an average of 5.4× and up to 23×. AHRC is effective: it obtained 20% higher F-measure than AHCKA, 24% higher ARI, 26% higher Jaccard Similarity, 10% higher Purity.
(4) AHRC intermediate graph representation $\mathbf{S}$ can be of general use. Notable improvements in clustering quality are observed by feeding $\mathbf{S}$ into the convolutional encoder of two cutting-edge attributed hypergraph contrastive learning models: averaged over tested real hypergraphs, by using $\mathbf{S}$, we outperform the best-in-class model TRICL by 38% in F-measure, 147% in ARI, 47% in Jaccard Similarity, and 22% in Purity.

The rest of this paper is organized as follows. Section 2 introduces the building blocks of our AHRC: attribute graph construction, hypergraph random walk, and modularity-based clustering. Section 3 presents our proposed attributed hypergraph representation approach. Section 5 describes the sparsification module for scalable computation and contrastive learning for the general use of the graph representation. Section 6 discusses the related work. Section 7 shows the empirical results. Section 8 concludes the paper.

2 Preliminary

Let A be a set of attributes. An *attributed hypergraph* $\mathcal{H}(V, E, \text{att})$ has a node set V, an edge set E where each edge $e \subseteq V$ is a subset

of V, and a function $\text{att} : V \mapsto 2^A$ that maps each node v in V to a subset $\text{att}(v) \subseteq A$ of attributes. For each node $v \in V$, define the degree $d_v(\mathcal{H})$ of v as the number of hyperedges in $\mathcal{H}$ that contain node v, i.e., $d_v(\mathcal{H}) = |\{e \in E | v \in e\}|$. For a set $C \subseteq V$ of nodes, denote by $\text{vol}_{\mathcal{H}}(C) = \sum_{v \in C} d_v(\mathcal{H})$ the volume of C. Denote by $\text{vol}(\mathcal{H}) = \text{vol}_{\mathcal{H}}(V)$ the volume of hypergraph $\mathcal{H}$. Denote by $n = |V|$ number of nodes in $\mathcal{H}$, $m = |E|$ the number of edges, $d = |A|$ the number of attributes. When $\mathcal{H}$ is clear in the context, we denote by d_v the degree of a node v and by $\text{vol}(C)$ the volume of a node set C. Denote by $\mathbf{H} \in R^{m \times n}$ the incident matrix of $\mathcal{H}$: for each edge e_i, $i \in [m]$ and each node v_j, $j \in [n]$, entry $\mathbf{H}[i, j] = [v_j \in e_i]$, i.e., $\mathbf{H}[i, j] = 1$ if v_j is incident to e_i and $\mathbf{H}[i, j] = 0$ if otherwise.

An attributed dyadic graph $G(V, E, \text{att})$ is a special attributed hypergraph where each edge $e \in E$ has exactly two nodes. Represent the graph G as an adjacency matrix $\mathbf{W}$: for two nodes v_i and v_j, $\forall i, j \in [n]$, $\mathbf{W}[i, j] = 1$ if there is an edge $(v_i, v_j) \in E$; otherwise $\mathbf{W}[i, j] = 0$. A weighted (dyadic) graph assigns a weight $w(e)$ to each edge $e \in E$, its adjacency matrix has $\mathbf{W}[i, j] = w(e)$ if $e(v_i, v_j) \in E$ and $\mathbf{W}[i, j] = 0$ if no edge in E connects v_i and v_j.

Clique Reduction [40]. Given an attributed hypergraph $\mathcal{H}(V, E, \text{att})$, clique reduction is a standard process that transforms $\mathcal{H}$ to an attributed dyadic graph $G_2(V, E_2, \text{att})$. Specifically, it converts each hyperedge $e \in E$ to a clique of nodes in e and unions the cliques to a dyadic graph G_2 with edge set $E_2 = \{(u, v) | \exists e \in E, s.t., u, v \in e\}$. We call $\text{vol}(G_2)$ the dyadic volume of $\mathcal{H}$ and denote it as $\text{vol}_2(\mathcal{H})$. The drawback of clique reduction is the loss of high-order information.

Property 1 (Attributed Hypergraph Clustering [46, 70]). *Given an attributed hypergraph $\mathcal{H}(V, E, \text{att})$, a clustering $\mathscr{C}$ of $\mathcal{H}$, a disjoint partitioning of V, is desirable if it satisfies two constraints: 1) nodes in the same cluster are closely connected to each other in terms of structure, while nodes between clusters are structurally separated, and 2) nodes in the same cluster have homogeneous attribute values, while nodes in different clusters may have diverse attribute values.*

Remarks. Property 1 shows the high-level objectives of existing AHC methods [46, 70]; however, their solutions assume that the number of clusters $|\mathscr{C}|$ in a desirable clustering is known in advance, which is not valid in reality. This paper focuses on the problem of finding a desirable AHC without a predefined cluster number.

2.1 Attribute Graph

To capture the attribute similarities among nodes, the techniques of K-Nearest Neighbor (KNN) search have been widely used [30, 46, 47]. Specifically, given an attributed hypergraph $\mathcal{H}(V, E, \text{att})$ and a parameter K, for each node $v \in V$, the K nodes $N_K(v_i)$ with the highest attribute similarity with v are computed. For two nodes $v_i, v_j \in V$, measure their attribute similarity with a cosine-similarity function $f(\text{att}(v_i), \text{att}(v_j))$ over their attribute sets. A straightforward similarity matrix can then be derived as follows:

$$\mathbf{M}[i, j] = \begin{cases} f(\text{att}(v_i), \text{att}(v_j)), & \text{if } v_j \in N_K(v_i) \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Because $\mathbf{M}$ is not symmetric, AHCKA [46] constructs a symmetric Attribute Similarity Matrix (ASM) $\mathbf{S_A} \in \mathbb{R}^{n \times n}$ by letting $\mathbf{S_A} = \mathbf{M} + \mathbf{M}^\intercal$ and uses $\mathbf{S_A}$ in the random walk for clustering. While computing the exact KNN graph could take quadratic time, fast approximate KNN algorithm [11, 27] has been adopted on large-scale attributed graphs due to its outstanding efficiency and accuracy. The attribute graph is a weighted graph constructed according to $\mathbf{S_A}$.

Example 1. *For example, in the hypergraph shown in Figure 1(a), node v_0 shares the same attribute "Blue" with nodes v_1 and v_2, yielding a cosine similarity of 1. In contrast, v_0 has a different attribute from nodes $v_3 - v_5$, their similarity is 0. As illustrated in Figure 2(a), when $K = 2$, both v_1 and v_2 are KNN neighbors of v_0. Similarly, v_0 is a KNN neighbor of v_1. So, $\mathbf{M}[0,1] = \mathbf{M}[1,0] = 1$ and $\mathbf{S_A}[0,1] = \mathbf{M}[0,1] + \mathbf{M}[1,0] = 2$.*

2.2 Hypergraph Random Walk

Random walk captures multi-hop similarities among nodes in a graph [31]. To preserve the high-order information in a hypergraph, the random walk is conducted in two steps [28]. Step 1 walks from a node v to an edge e chosen uniformly at random from all the incident hyperedges of v. Step 2 walks from e to a node u chosen uniformly at random from all the nodes in e. Formally, given an attributed hypergraph $\mathcal{H}$ with incident matrix $\mathbf{H} \in \mathbb{R}^{m \times n}$, let $\mathbf{T_V} \in \mathbb{R}^{n \times m}$ and $\mathbf{T_E} \in \mathbb{R}^{m \times n}$ be the row-normalized matrices of $\mathbf{H}^\intercal$ and $\mathbf{H}$, respectively. The transition matrix of Step 1 is $\mathbf{T_V}$ and that of Step 2 is $\mathbf{T_E}$. We call $\mathbf{T} = \mathbf{T_V} \times \mathbf{T_E}$ the **hypergraph transition matrix**.

Based on the hypergraph transition matrix defined above, the random walk with restart process on dyadic graph [62] can be generalized to hypergraph, to capture the multi-hop topology in the hypergraph. Formally, from a node $u \in V$, an α, γ-**Hypergraph Random Walk** moves in γ steps where in each step:

- With probability α, terminates at the current node and then jumps back to the source node u;
- With probability $1 - \alpha$, transits from the current node v_i to a node v_j based on the hypergraph transition matrix $\mathbf{T}$.

2.3 Modularity-based Clustering

On dyadic graphs, a widely adopted line of clustering optimizes the modularity function proposed by Newman-Girvan [51]. Given an unweighted dyadic graph $G(V, E)$ and a random graph model [2] that preserves the degree distribution of G, the NG modularity $\text{NG}(C)$ of a subset C of nodes in G is defined as follows.

$$\text{NG}(C) = \frac{|E(C)| - Exp[|E(C)|]}{m} = \frac{2|E(C)|}{\text{vol}(G)} - \left(\frac{\text{vol}(C)}{\text{vol}(G)}\right)^2 \quad (2)$$

where $E(C) = \{(u, v) \in E | u, v \in C\}$ is the set of edges with both ends in C. For a clustering $\mathscr{C}$, the modularity for $\mathscr{C}$ is the sum of modularity for each cluster $C \in \mathscr{C}$, i.e., $\text{NG}(\mathscr{C}) = \sum_{C \in \mathscr{C}} \text{NG}(C)$. The NG modularity measures the difference between the actual number of innercluster edges of G and the expected number of innercluster edges of a random graph. A higher modularity score indicates a more pronounced clustering structure: nodes within the same cluster of $\mathscr{C}$ are more closely connected in G than that would be anticipated in a random graph.

Modularity-based clustering is highly popular [5, 14, 22, 33, 52] especially in large-scale graph applications because it requires no prior knowledge of the cluster number, i.e., it decides the cluster number automatically, and moreover, its algorithm, e.g., Louvain [5], achieves both high scalability and clustering quality [77].

Our proposed clustering method is established based on the above building blocks, which will be introduced in Section 3.

3 Clustering Attributed Hypergraph

In this section, we introduce the backbone (modules shaded in Figure 1) of the hypergraph clustering pipeline of Attributed Hypergraph Representation for Clustering (AHRC) in two parts. Section 3.1 elaborates attributed hypergraph representation (AHR) which integrates hypergraph topology and attribute information into an Integrated Similarity Matrix (ISM) $\mathbf{S}$. Section 3.2 formulates, based on $\mathbf{S}$, an integrated multi-hop modularity, as the objective function for modularity-based clustering.

3.1 Attributed Hypergraph Representation

The topological similarity between nodes in a graph is computed based on α, γ-Hypergraph Random Walk introduced in Section 2, which derives the Topology Similarity Matrix (TSM) $\mathbf{S_T} \in \mathbb{R}^{n \times n}$

$$\mathbf{S_T} = \alpha \sum_{l=0}^{\gamma} (1-\alpha)^l \mathbf{T}^l, \tag{3}$$

where entry $\mathbf{S_T}[i,j]$ is the probability that an α,γ-hypergraph random walk from v_i terminates at v_j under hypergraph transition matrix $\mathbf{T}$ defined in Section 2.2. $\mathbf{S_T}$ captures multi-hop topological similarity by considering random walks up-to-γ lengths. Specifically, α controls the probability of restarting the random walk from the initial node at each step, balancing local and global topological information. $\mathbf{T}^l$ represents the probability of transitioning from one node to another in exactly l steps. The summation $\sum_{l=0}^{\gamma}(1-\alpha)^l\mathbf{T}^l$ captures the contribution of walks of different lengths (from 0 to γ hops) to the overall topological similarity.

The parameter γ can be infinite, but it is practically set to a constant for an efficient approximation [46]. Our empirical studies suggest that $\gamma = 2$ strikes a balance between the computation cost and effectiveness and thus is set as a default value. Lemma 1 shows the computational time and space complexities of $\mathbf{S_T}$ when $\gamma = 2$. The proof of Lemma 1 indicates that the main cost in computing $\mathbf{S_T}$ arises from the large dyadic volume $\mathrm{vol}_2(\mathcal{H})$. To mitigate this issue, Section 4 will show a sparsification process to reduce $\mathrm{vol}_2(\mathcal{H})$.

Lemma 1. *Given a hypergraph $\mathcal{H}$ and let $\gamma = 2$, the computation of $\mathbf{S_T}$ takes $O(\frac{\mathrm{vol}_2(\mathcal{H})^2}{n})$ time and $O(\frac{\mathrm{vol}_2(\mathcal{H})^2}{n})$ memory space in the average case.*

Proof. Given a hypergraph $\mathcal{H}$, the number of non-zero entries in the transition matrix $\mathbf{T}$ is $O(\mathrm{vol}_2(\mathcal{H}))$ because any two nodes have non-zero transition probability if they have at least one common incident hyperedge. Since $\mathbf{T}$ is a sparse matrix, the time complexity of computing matrix power $\mathbf{T}^2$ is $O(\frac{\mathrm{vol}_2(\mathcal{H})^2}{n})$ [73]. Since the sparse matrix power takes the main computational cost, the overall time complexity of computing $\mathbf{S_T}$ is thus to be $O(\frac{\mathrm{vol}_2(\mathcal{H})^2}{n})$. The space overhead is also determined by the densest matrix $\mathbf{T}^2$. For nodes $v_i, v_j \in V$, we call v_j a 1-hop neighbor of v_i if entry $\mathbf{T}[i,j] > 0$. As there are $O(\mathrm{vol}_2(\mathcal{H}))$ non-zero entries in $\mathbf{T}$, the average number of 1-hop neighbors of a node is $O(\frac{\mathrm{vol}_2(\mathcal{H})}{n})$. Similarly, we call node v_j to be the 2-hop neighbor of v_i if $\mathbf{T}^2[i,j] > 0$. The average number of 2-hop neighbors of a node is thus expected to be $O((\frac{\mathrm{vol}_2(\mathcal{H})}{n})^2)$. Thus, summing up over n nodes, the overall space complexity is expected to be $O(\frac{\mathrm{vol}_2(\mathcal{H})^2}{n})$ in the average case. □

Algorithm 1: Integrator

Input: Topological similarity matrix $\mathbf{S_T}$, attributed similarity matrix $\mathbf{S_A}$
Output: Integrated similarity matrix $\mathbf{S}$
1 Compute row normalization matrices $\mathbf{S_T} \leftarrow \mathrm{norm}(\mathbf{S_T})$ and $\mathbf{S_A} \leftarrow \mathrm{norm}(\mathbf{S_A})$;
2 Compute $\mathbf{S'} \leftarrow \mathbf{S_T} \times \mathbf{S_A}$;
3 **for** each entry $\mathbf{S'}[i,j]$ **do** Let $\mathbf{S}[i,j] \leftarrow \sqrt{\mathbf{S'}[i,j]}$;
4 **return** $\mathbf{S}$;

On the other hand, we can employ a fast approximate KNN search to construct the attribute graph and the corresponding Attribute Similarity Matrix (ASM) $\mathbf{S_A}$ based on Section 2.1.

Integrated Similarity. With Topology Similarity Matrix (TSM) and Attribute Similarity Matrix (ASM), we now compute the integrated similarity between nodes. We first show the steps of the integration and then elaborate on the rationales behind the integration.

Algorithm 1 shows the pseudo code for computing the integrated similarity. Given the topological similarity matrix $\mathbf{S_T}$ and attribute similarity matrix $\mathbf{S_A}$, Line 1 performs row normalizations on both $\mathbf{S_T}$ and $\mathbf{S_A}$ to convert each row into probability distributions. Consequently, entry $\mathbf{S_T}[i,j]$ (resp. $\mathbf{S_A}[i,j]$) denotes the topological (resp. attributed) similarity of v_j from the perspective of v_i. Line 2 defines $\mathbf{S'} \in \mathbb{R}^{n \times n}$ as $\mathbf{S'} = \mathbf{S_T} \times \mathbf{S_A}$ where entry $\mathbf{S'}[i,j] = \sum_{v_r \in V} \mathbf{S_T}[i,r] \cdot \mathbf{S_A}[r,j]$ is the weighted sum of the product of $\mathbf{S_T}[i,r]$ and $\mathbf{S_A}[r,j]$ over all intermediate nodes v_r. Line 3 transforms $\mathbf{S'}$ to Integrated Similarity Matrix (ISM) $\mathbf{S}$ by applying a square root transformation [54], i.e., $\mathbf{S}[i,j] = \rho(\mathbf{S'}[i,j]) = \sqrt{\mathbf{S'}[i,j]}$, for each pair $i, j \in [n]$.

Example 2. *Given row-normalized $\mathbf{S_T}$ and $\mathbf{S_A}$, Figure 2(c) shows how to compute the entry $\mathbf{S}[2,1]$. First, we calculate the dot product of $\mathbf{S_T}[2,:]$ and $\mathbf{S_A}[:,1]$ as $\mathbf{S'}[2,1] = \mathbf{S_T}[2,:] \cdot \mathbf{S_A}[:,1] = 0.45$. Then, the square root transformation is applied, giving $\mathbf{S}[2,1] = \sqrt{0.45} \approx 0.7$.*

Interpretation. $\mathbf{S'}$ propagates the multi-hop topological similarity across the attribute graph. Specifically, for two nodes v_i, v_j, and an intermediate node v_r, the topological similarity between v_i and v_r is passed on to v_j if v_j has a similar set of attributes with v_r. In other words, if v_j and v_r are similar by nature (attribute-wise), they exchange the information of their topological neighbors in the computation of $\mathbf{S'}$. $\mathbf{S'}[i,j]$ reflects a similarity between v_i and v_j in terms of both topology and attributes. Integrated Similarity Matrix (ISM) $\mathbf{S}$ is eventually computed by applying a square root transformation on $\mathbf{S'}$ for a better similarity distribution. Specifically, in the presence of unbalanced graph structures [76], existing optimization methods adopting objectives (e.g., cut ratio [42], conductance [49] and modularity [23]) empirically does not perform well. In other words, they tend to favor graphs with balanced ground truth clusterings (i.e., each cluster has similar volume). By applying the concave square root function, large values become less influential, leading to a more even distribution of similarities and consequently, a better clustering quality [24]. The choice of the smooth function is not

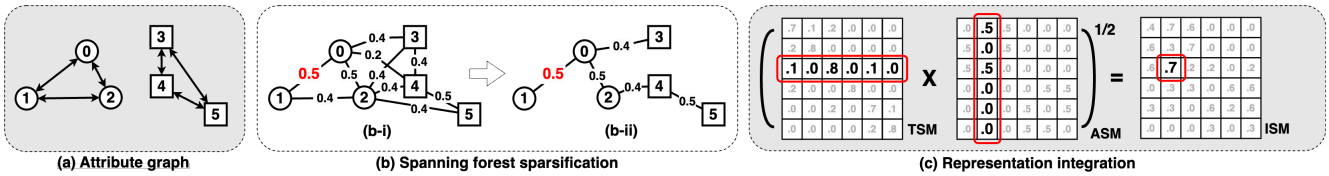
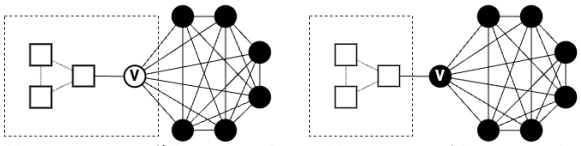


Figure 2: Illustrative examples of our pipeline AHRC



(1) Bad clustering $\mathscr{C}'$ (0.34 → 0.25) (2) Clustering $\mathscr{C}$ (0.32 → 0.27)

Figure 3: Example on how transformation affects modularity

exclusive; we conducted experiments (Exp 6 in Section 7) which suggests square root function is an ideal candidate.

Example 3. *To illustrate the impact of the square root transformation on modularity-based clustering, consider Figure 3. The graph consists of a 3-clique and a 7-clique, connected by a single edge. Naturally, each clique would be its own cluster. Without the transformation, node v might be incorrectly assigned to the 3-clique cluster because the clustering $\mathscr{C}'$ in Figure 3 (1) has a higher modularity* (0.34 > 0.32). *However, after applying the square root transformation, the modularity of $\mathscr{C}'$ decreases significantly to* 0.25, *making it lower than that of the true clustering of $\mathscr{C} = \{3\text{-}clique, 7\text{-}clique\}$ in Figure 3 (2).*

Lemma 2 shows the time and space complexities of Algorithm 1.

Lemma 2. *Given a topological similarity matrix $\boldsymbol{S_T}$ and an attribute similarity matrix $\boldsymbol{S_A}$, Algorithm 1 takes $O(\frac{K\cdot \mathrm{vol}_2(\mathcal{H})^2}{n})$ time and $O(\frac{K\cdot \mathrm{vol}_2(\mathcal{H})^2}{n})$ memory space.*

Proof. We first prove that the space complexity of $\mathbf{S}$ is $O(K \cdot \mathrm{vol}_2(\mathcal{H})^2)$, where K is the parameter for the KNN algorithm. Since the average number of non-zero entries per row in sparse matrices $\mathbf{S_T}$ and $\mathbf{S_A}$ is $\frac{\mathrm{vol}_2(\mathcal{H})^2}{n^2}$ and K, respectively, a node can access $O(\frac{K\cdot \mathrm{vol}_2(\mathcal{H})^2}{n^2})$ number of nodes on $\mathbf{S_T} \times \mathbf{S_A}$. The number of non-zero entries in $\mathbf{S}$ (the memory cost) is thus $O(\frac{K\cdot \mathrm{vol}_2(\mathcal{H})^2}{n})$.

Then, we prove the time complexity. Line 1 normalizes matrices $\mathbf{S_T}$ and $\mathbf{S_A}$, taking $O(\frac{\mathrm{vol}_2(\mathcal{H})^2}{n} + K\cdot n)$ time. Since both $\mathbf{S_T}$ and $\mathbf{S_A}$ are sparse matrices with $O(\frac{\mathrm{vol}_2(\mathcal{H})^2}{n})$ and $O(Kn)$ numbers of non-zero entries, respectively, the complexity of Line 2 is $O(\frac{K\cdot \mathrm{vol}_2(\mathcal{H})^2}{n})$ [73]. Line 3 performs a transformation on $\mathbf{S}$, taking $O(\frac{K\cdot \mathrm{vol}_2(\mathcal{H})^2}{n})$ time. Overall, Algorithm 1 takes $O(\frac{K\cdot \mathrm{vol}_2(\mathcal{H})^2}{n})$ time. □

Remarks. We represent the attributed hypergraph with $\mathbf{S}$ which combines both graph topology and attribute information. Specifically, consider two nodes v_i and v_j. If v_i and v_j are closely connected topologically and share homogeneous attributes, the probability of a random walk connecting them should be high, leading to a large value of $\mathbf{S}[i, j]$. Conversely, if v_i and v_j are distant with dissimilar attributes, and there is no node that is similar to v_j (in terms of attributes) and topplogically close to v_i, the value of $\mathbf{S}[i, j]$ should be small. Section 3.2 defines an integrated multi-hop modularity for clustering.

3.2 Integrated Multi-hop Modularity

The definition of NG modularity fails to capture the constraints of AHC (as described in Property 1), as it considers neither multi-hop topology nor attribute information. To address this issue, we propose an objective function, called Integrated Multi-Hop Modularity (IMM). Specifically, given an attributed hypergraph $\mathcal{H}$, Section 3.1 computes a similarity matrix $\mathbf{S}$ using our proposed attributed hypergraph representation (AHR). Regard $\mathbf{S}$ as the adjacency matrix of a weighted dyadic graph where entry $\mathbf{S}[i, j] = 0$ indicates there is no edge between nodes v_i and v_j. We call this weighted dyadic graph the *representative graph* of $\mathcal{H}$, on which we define the integrated multi-hop modularity, denoted as IMM, as follows.

Definition 1 (Integrated Multi-Hop Modularity). *Given an attributed hypergraph $\mathcal{H}$, a clustering $\mathscr{C}$, and a similarity matrix $\boldsymbol{S}$ under the* AHR *model, the integrated multi-hop modularity of the clustering $\mathscr{C}$ is defined as:*

$$\mathrm{IMM}(\mathscr{C}) = \sum_{C\in\mathscr{C}} \frac{\sum_{v_i,v_j\in C} \boldsymbol{S}[i,j]}{\sum_{v_i,v_j\in V} \boldsymbol{S}[i,j]} - \left(\frac{\sum_{v_i\in C, v_j\in V} \boldsymbol{S}[i,j]}{\sum_{v_i,v_j\in V} \boldsymbol{S}[i,j]}\right)^2. \quad (4)$$

Note that $\sum_{v_i,v_j\in C} \mathbf{S}[i, j]$ is the sum of similarities between all pairs of nodes within cluster C, and $\sum_{v_i\in C, v_j\in V} \mathbf{S}[i, j]$ is the sum of similarities of all nodes in C with its neighbors. Recall that the IMM does not require a predefined cluster number k. The subsequent process is to partition all nodes within G, aiming to find the clustering $\mathscr{C} = \{C_1, C_2, \cdots, C_{|\mathscr{C}|}\}$ such that their IMM score is maximized. Exact modularity optimization is NP-hard [8], leading to approximation approaches such as Louvain [5].

Remarks. Different from the classic NG modularity function, which merely relies on edges (1-hop relation), the IMM function (Definition 1) captures multi-hop relations under our AHR model, which encodes high-order information in $\mathbf{H}$. Specifically, the random walk takes into account the paths that start from and end at nodes within the same cluster: given a cluster C, IMM computes the difference between the actual possibility that paths on the data graph stay within C and, the expected possibility that the paths on the random graph stay within C. A higher modularity score indicates that the actual paths within clusters are (probabilistically) more than what would be expected in a random graph, suggesting a good clustering structure. Paths can capture higher-order relationships between nodes, providing a richer representation of connectivity in the data graphs. Implicitly generalizing modularity from edges to paths 1) allows the extraction of more informative features on the data graphs, 2) is an alternative approach to overcome the resolution limit of modularity (struggling to identify small clusters) [18], and thus 3) often brings higher clustering quality [25].

Algorithm 2: AHRC

Input: Attributed hypergraph $\mathcal{H}(V, E, \text{att})$, attribute similarity matrix $\mathbf{S_A}$, decay factor α, number of iteration γ, sparsification parameter τ, and boolean $spax$: switch of the sparsification

Output: Clustering $\mathscr{C}$

1 $\mathbf{H} \leftarrow$ incident matrix of $\mathcal{H}$;
2 Compute row normalization matrices $\mathbf{T_V} \leftarrow \text{norm}(\mathbf{H}^\intercal)$ and $\mathbf{T_E} \leftarrow \text{norm}(\mathbf{H})$;
3 Compute transition matrix $\mathbf{T} \leftarrow \mathbf{T_V} \times \mathbf{T_E}$;
4 **if** $spax$ **then**
5 | Perform matrix sparsification $\mathbf{T} \leftarrow \text{Sparsifier}(\mathbf{T}, \tau)$;
6 Compute topological similarity matrix $\mathbf{S_T} \leftarrow \alpha \sum_{l=0}^{\gamma} (1-\alpha)^l \mathbf{T}^l$;
7 Compute integrated similarity matrix $\mathbf{S} \leftarrow \text{Integrator}(\mathbf{S_T}, \mathbf{S_A})$;
8 $\mathscr{C} \leftarrow \text{Louvain}(\mathbf{S})$;
9 **return** $\mathscr{C}$;

3.3 The AHRC Algorithm

The process of AHRC starts with an attributed hypergraph as input, computes the integrated similarity matrix as representation to capture both graph topological and attribute information using the AHR model, and performs clustering based on this representation.

Algorithm 2 shows the pseudo code of the AHRC. It takes as input an attributed hypergraph $\mathcal{H}(V, E, \text{att})$, the attribute similarity matrix $\mathbf{S_A}$ of $\mathcal{H}$, a decay factor α, and the number of iteration γ for hypergraph random walk. Two additional inputs, a sparsification parameter τ and a boolean indicator $spax$ will also be taken when AHRC applies the proposed spanning forest sparsification process (will be described in Section 4). In Line 1, AHRC first extracts the incident matrix $\mathbf{H}$ of $\mathcal{H}$ followed by computing the row normalization matrices $\mathbf{T_V}$ and $\mathbf{T_E}$ in Line 2. Then, the transition matrix $\mathbf{T}$ of α, γ-Hypergraph Random Walk is computed (Line 3). After that, in Line 6, AHRC computes the TSM $\mathbf{S_T}$ according to Equation 3. Algorithm 1 is then called in Line 7 to compute the ISM $\mathbf{S}$. Based on the obtained $\mathbf{S}$, the Louvain method is then applied to do the clustering (Line 8). Line 9 returns the resulting clustering $\mathscr{C}$. Lemma 3 analyzes the time complexity of the AHRC algorithm without spanning forest sparsification.

Lemma 3. *When $\gamma = 2$, the time complexity of Algorithm 2 without spanning forest sparsification is $O(\frac{K \cdot \text{vol}_2(\mathcal{H})^2}{n})$.*

Proof. Since the number of non-zero entries in sparse matrix $\mathbf{H}$ is $\text{vol}(\mathcal{H})$, Line 2 takes $O(\text{vol}(\mathcal{H}))$ time. Both sparse matrices $\mathbf{T_V} \in \mathbb{R}^{n \times m}$ and $\mathbf{T_E} \in \mathbb{R}^{m \times n}$ have $\text{vol}(\mathcal{H})$ non-zero entries, their multiplication takes $O(\frac{\text{vol}(\mathcal{H})^2}{n})$ time [73] in Line 3. According to Lemma 1, Line 6 takes $O(\frac{\text{vol}_2(\mathcal{H})^2}{n})$ time. Line 7 then calls Algorithm 1 to multiply matrices $\mathbf{S_T}$ and $\mathbf{S_A}$, taking $O(\frac{K \cdot \text{vol}_2(\mathcal{H})^2}{n})$ time according to Lemma 2. Since the matrix $\mathbf{S}$ has $O(\frac{K \cdot \text{vol}_2(\mathcal{H})^2}{n})$ non-zero entries, the Louvain method called in Line 8 is thus to be $O(\frac{K \cdot \text{vol}_2(\mathcal{H})^2}{n})$. Therefore, the total time complexity of the AHRC algorithm is $O(\frac{K \cdot \text{vol}_2(\mathcal{H})^2}{n})$. □

4 Spanning Forest Sparsification

This section introduces the module of spanning forest sparsification (Figure 1) which addresses the main scalability bottleneck of AHRC. Lemma 3 indicates that the main scalability bottleneck of AHRC is the large dyadic volume $\text{vol}_2(\mathcal{H})$ resulting from the dense hypergraph transition matrix $\mathbf{T}$. It's a natural idea to consider how to reduce $\text{vol}_2(\mathcal{H})$ through sparsification. In this section, we propose a linear-time graph sparsification method called Spanning Forest Sparsification. Recall the definition of $\mathbf{T}$ and hypergraph random walk in Section 2.2, we have a Lemma as follows.

Lemma 4. *Given a hypergraph transition matrix $\boldsymbol{T}$, define a binary matrix of $\boldsymbol{B_T}$ such that $\boldsymbol{B_T}[i,j] = 1$ if $\boldsymbol{T}[i,j] > 0$ and $\boldsymbol{B_T}[i,j] = 0$ otherwise. Then, $\boldsymbol{B_T}$ is a symmetric matrix.*

Proof. To prove that $\mathbf{B_T}$ is a symmetric matrix, we show that $\mathbf{B_T}[i,j] = \mathbf{B_T}[j,i]$ for every i, j. For any pair of nodes v_i and v_j, assume $\mathbf{T}[i,j] > 0$. It implies that v_i can transit to v_j through a one-step random walk. According to the definition of hypergraph random walk, v_i can transit to v_j if and only if they share at least one incident hyperedge. Thus, v_i and v_j share at least one incident hyperedge, and through this shared hyperedge, v_j can also transit to v_i, leading to $\mathbf{T}[j,i] > 0$. Therefore, we have $\mathbf{T}[j,i] > 0$ if $\mathbf{T}[i,j] > 0$. We can prove that $\mathbf{T}[i,j] > 0$ if $\mathbf{T}[j,i] > 0$ similarly by reversing the roles of v_i and v_j. Thus, for every i, j, we have $\mathbf{T}[j,i] > 0$ if and only if $\mathbf{T}[i,j] > 0$, and therefore $\mathbf{B_T}[j,i] = \mathbf{B_T}[i,j]$. □

In this section, we represent $\mathbf{T}$, and hence the random walk, as a weighted directed graph [28] G with edge set $\{e(i,j) | \mathbf{T}[i,j] > 0\}$, where the edge weight of $e(i,j)$ is the transition probability from node v_i to v_j. According to Lemma 4, G has an undirected structure: for any node pairs (v_i, v_j), there is an edge from v_i to v_j if and only if there is another edge from v_j to v_i. G has asymmetric weights: due to the asymmetry of the hypergraph random walk, the weights of edge $e(v_i, v_j)$ may not equal to that of $e(v_j, v_i)$.

Given a graph G with aforementioned properties, our method constructs the union of a set of edge-disjoint maximum spanning forests, resulting in a sparsified graph that 1) is a structurally connected subgraph of G, 2) maintains the asymmetric edges weights of G, and 3) preserve significant relationships carrying large edge weights.

Given a transition matrix $\mathbf{T}$ and a parameter τ, we first compute the symmetric matrix $\mathbf{T} + \mathbf{T}^\intercal$, combining transition probabilities in both directions between nodes, yielding an adjacency matrix of an undirected graph, denoted as G_{sym}. This symmetrization allows us to apply Kruskal's algorithm [37], which is designed to find the maximum spanning tree (forest) on undirected graphs. Next, on G_{sym}, we generate the union of a set of edge-disjoint maximum spanning forests $F_1, F_2, \cdots, F_\tau$. Each F_i is a maximum spanning forest on G_{sym} after removing those edges in $F_1, F_2, \cdots, F_{i-1}$. The union of these forests is denoted as $F = \cup_{i \in [\tau]} F_i$. F can be computed iteratively. Specifically, we maintain two graphs: a residual graph G^- and a cumulative graph G^+. Initially, G^- is the same as G_{sym} and G^+ has the same node set as G_{sym} but starts with an empty edge set. In each iteration i, we generate a maximum spanning forest F_i for G^- by Kruskal's algorithm. We then update G^- by subtracting the set of edges in F_i from G^-, and update G^+ by taking the union of the edge sets of F_i and G^+. The process repeats τ times.

Algorithm 3: Sparsifier

Input: Transition matrix $\mathbf{T}$ and sparsification parameter τ
Output: Sparsified transition matrix $\mathbf{T}'$

1 Initialize cumulative matrix $\mathbf{M}^+ \leftarrow \mathbf{0}$;
2 Initialize residual matrix $\mathbf{M}^- \leftarrow \mathbf{T} + \mathbf{T}^\intercal$;
3 **for** $i \leftarrow 1$ *to* τ **do**
4 Find maximum spanning forest $\mathbf{F_i} \leftarrow$ Kruskal($\mathbf{M}^-$);
5 Update $\mathbf{M}^- \leftarrow \mathbf{M}^- - \mathbf{F_i}$ and $\mathbf{M}^+ \leftarrow \mathbf{M}^+ + \mathbf{F_i}$;
6 $\mathbf{T}' \leftarrow \mathbf{0}$;
7 **for** each non-zero entry $\mathbf{M}^+[i,j]$ **then** $\mathbf{T}'[i,j] \leftarrow \mathbf{T}[i,j]$;
8 **return** $\mathbf{T}'$;

After termination, the edge set of G^+ is the edge set of F. Finally, for each undirected edge $e(i,j)$ in F, we retain a pair of directed edges $e(i,j)$ and $e(j,i)$ in the sparsified transition matrix $\mathbf{T}'$.

Example 4. *Consider the hypergraph in Figure 1(a) where the asymmetric transition probabilities between v_0 and v_1 is $\mathbf{T}[0,1] = 0.2$ and $\mathbf{T}[1,0] = 0.3$, respectively. In Figure 2(b-i), we first compute the symmetric matrix of $\mathbf{T}$, yielding an undirected graph where edge $e(v_0, v_1)$ has a weight of $0.2 + 0.3 = 0.5$. Figure 2(b-ii) then shows the spanning tree on the graph, preserving connectivity and prioritizing larger edge weights. The asymmetric transition probability $\mathbf{T}[0,1]$ and $\mathbf{T}[1,0]$ can then be recovered from the tree.*

Algorithm 3 presents the pseudo code of the sparsification method, where the residual graph G^-, cumulative graph G^+, and maximum spanning forest F_i is represented by the adjacency matrix $\mathbf{M}^-$, $\mathbf{M}^+$, and $\mathbf{F_i}$, respectively. Algorithm 3 takes the transition matrix $\mathbf{T}$ and a parameter τ as inputs. In Line 1, the matrix $\mathbf{M}^+$ is initialized as a zero matrix with the same shape as $\mathbf{T}$. In Line 2, the matrix $\mathbf{M}^-$ is initialized to be $\mathbf{T} + \mathbf{T}^\intercal$. Lines 3-5 iteratively find the maximum spanning forest $\mathbf{F_i}$ by calling the Kruskal's algorithm and update the matrices $\mathbf{M}^-$ and $\mathbf{M}^+$. Lemma 5 analyzes the time complexity of the AHRC algorithm with sparsification.

Lemma 5. *When $\gamma = 2$, the time complexity of Algorithm 2 using the spanning forest sparsification method is $\tilde{O}(\mathrm{vol}_2(\mathcal{H}))$.*

Proof. Since the transition matrix $\mathbf{T}$ has $O(\mathrm{vol}_2(\mathcal{H}))$ number of non-zero entries according to the proof of Lemma 1, Algorithm 3 takes $O(\tau \cdot \mathrm{vol}_2(\mathcal{H}) \log n)$ time [37] for performing the sparsification. Therefore, in Algorithm 2, Line 4-5 takes $O(\tau \cdot \mathrm{vol}_2(\mathcal{H}) \log n)$ time by calling Algorithm 3. Given that τ forests have $O(\tau n)$ edges in total, the sparsified transition matrix has $O(\tau n)$ number of non-zero entries. According to Lemma 1, Line 6 of Algorithm 2 then takes $O(\tau^2 n)$ time to compute matrix $\mathbf{S_T}$ with $O(\tau n)$ non-zero entries. Line 7 multiplies matrices $\mathbf{S_T}$ and $\mathbf{S_A}$, taking $O(\tau K n)$ time according to Lemma 2. The resulting matrix $\mathbf{S}$ has $O(\tau K n)$ non-zero entries. The Louvain algorithm called in Line 8 thus takes $O(\tau K n)$ time. Therefore, the overall time complexity of Algorithm 2 is $O(\tau \cdot \mathrm{vol}_2(\mathcal{H}) \log n + (\tau + K)\tau n)$. Assuming τ and K are small constants, the overall time complexity of AHRC with sparsification is $\tilde{O}(\mathrm{vol}_2(\mathcal{H}))$. □

Limitation. The proof of Lemma 5 shows that the spanning forest sparsification reduces the dyadic volume $\mathrm{vol}_2(\mathcal{H})$ of a hypergraph for clustering, which brings dramatic improvement in the clustering computation. In the spanning forest sparsification itself, however, AHRC still faces a limitation as a main-memory algorithm. In other words, if the $\mathrm{vol}_2(\mathcal{H})$ of the original hypergraph exceeds the memory limit, $\mathbf{T}$ cannot be computed in main memory. A possible remedy to this limitation is to resort to external memory spanning forest sparsification, which has been listed as our future work. According to the proof of Lemma 5, with $\mathbf{T}$ computed, the computational complexity of the remaining processes in AHRC becomes $O((\tau + K)\tau n)$ when $\gamma = 2$, thereby bypassing the dependence on $\mathrm{vol}_2(\mathcal{H})$.

5 Enhancing Contrastive Learning with AHRC

This section introduces the module of contrastive learning as a general application of our graph representation $\mathbf{S}$. In the context of GNN-based attributed graph clustering, Graph Contrastive Learning (GCL) [72] has emerged as a popular framework. It learns an encoding function that takes node attributes and graph topology as input, and produces node embeddings as output. These embeddings can then be used for clustering by applying k-Means algorithm. In this section, we show how the attributed hypergraph representation generated by our AHR model enhances existing GCL methods.

We proposed two models TCL+ and GRC+, as enhanced variants of the state-of-the-art GCL methods TRICL [43] and GRACE [81], respectively, based on our AHRC. Figure 4 elaborates the module of contrastive learning + k-means in Figure 1 when it comes to TCL+. It consists of four major components: graph augmentation, encoder, projection head, and contrastive loss. Typically, the encoder is composed of one or more graph neural network layers built on the underlying graph structure. Under our AHRC, we propose an AHR layer that enriches embeddings with the comprehensive multi-hop topological and attribute information captured by the AHRC.

AHR **Layer.** The AHR layer is built on the Integrated Similarity Matrix (ISM) $\mathbf{S}$ under our AHRC pipeline. Given an attributed hypergraph, we first compute ISM $\mathbf{S}$ using AHRC. Regarding $\mathbf{S}$ as a weighted directed graph $G(V, E)$, graph convolution is then applied to the underlying unweighted structure of G to generate node embeddings for the contrastive learning process. Specifically, the AHR Layer iteratively propagates embeddings through the unweighted structure of G, updating the embedding of each node by aggregating the embedding of its adjacent nodes. Let $\mathbf{z}_v^{(i)}$ be the embedding of node $v \in V$ at the i-th AHR layer, defined as:

$$\mathbf{z}_v^{(i)} = f(\mathbf{z}_v^{(i-1)}, \{\mathbf{z}_u^{(i-1)} : e(u,v) \in E\}) \tag{5}$$

where f is the aggregation rule. Then, our proposed models, TCL+ and GRC+, incorporate the AHR layers to the encoders of TRICL and GRACE, respectively.

TCL+ **Model Architecture.** Figure 4 overviews the architecture of our proposed model TCL+. We briefly introduce it in the following.

(1) **Graph augmentation.** Given an attributed hypergraph $\mathcal{H}$, we first compute an ISM $\mathbf{S}$ using the AHRC pipeline. Next, we augment $\mathbf{S}$ by performing random *edge removing* [81], which we refer to as AHR edge removing, to generate two alternate views on $\mathbf{S}$. In these views, a portion (controlled by a hyperparameter p_d) of non-zero entries in $\mathbf{S}$ are randomly set to be

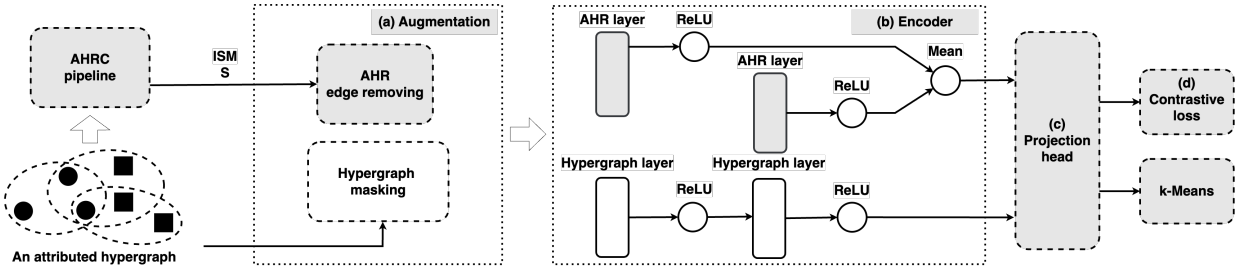


Figure 4: TCL+: AHRC **for Contrastive Learning based Clustering**

zero. Additionally, we perform hypergraph masking to augment the hypergraph topology and attributes by performing random *membership masking* [43] and *node feature masking* [81] to generate two alternate views of the hypergraph.

(2) **Encoder.** The encoder produces embeddings for the views generated in (1). As Figure 4(b) shows, TCL+ employs a two-level encoder, each of which consists of one AHR layer and one hypergraph layer. All layers use the element-wise mean pooling aggregation rule as a special instance of Equation 5. Specifically, the AHR layer can be represented in the matrix form:

$$\mathbf{Z}^{(i)} = \sigma(\mathbf{D}^{-1}\mathbf{B}\mathbf{Z}^{(i-1)}\mathbf{W}^{(i)}) \tag{6}$$

where $\mathbf{B}$ is a binary adjacency matrix such that $\mathbf{B}[i,j] = 1$ if $\mathbf{S}[i,j] > 0$ and $\mathbf{B}[i,j] = 0$ otherwise, representing the unweighted structure of G. The initial node embeddings $\mathbf{Z}^{(0)}$ are set to the node attributes. $\mathbf{D}$ is the diagonal degree matrix where $\mathbf{D}[i,i] = \sum_j \mathbf{B}[i,j]$, $\mathbf{W}^{(i)}$ is the trainable weight for the i-th layer, and σ is the activation function $\mathrm{ReLU}(x) = \max(0,x)$. The hypergraph layer applied on the hypergraph topology follows the same structure as that of [43].

(3) **Projection head and contrastive loss.** With the embeddings from encoder, we project them by performing non-linear transformation [12] using the same projection heads as [43]. For node embeddings generated by both hypergraph layers and AHR layers, we adopt the same objective function as [43, 81]. The overall loss is computed as follows:

$$\mathcal{L}_n = \mathcal{L}_H + w_s \cdot \mathcal{L}_A \tag{7}$$

where $\mathcal{L}_H$ and $\mathcal{L}_A$ is the contrastive loss on the node embeddings generated by the hypergraph layers and AHR layers, respectively. w_s is the weight balancing two losses.

GRC+ **Model Architecture.** GRACE is a state-of-the-art GCL method on dyadic graphs, employing a two-layer encoder. On hypergraphs, GRACE can be applied on a dyadic graph G' that is converted from a given attributed hypergraph through clique reduction. Our proposed model, GRC+, enhances GRACE by replacing the layers built on G' with the AHR layers built on the ISM $\mathbf{S}$ under our AHC model.

6 Related Works

Attributed Hypergraph Clustering. Graph clustering [23, 56] has been extensively studied on dyadic graphs. Traditional clustering optimizes objective functions such as modularity [15], conductance [6], normalized cut [58], etc. Exact modularity optimization is computationally hard, leading to approximation approaches [5, 16, 20, 52]. Among these, Louvain [5] has been widely used in industry due to its scalability and clustering quality [77]. A hypergraph can be transformed into a dyadic graph using clique reduction [1, 41], and then dyadic graph clustering methods can be applied. However, this approach loses high-order information in hypergraphs. Other methods represent a hypergraph using a hypergraph random walk transition matrix [28, 78] or a normalized Laplacian [44, 45], transforming it into a weighted dyadic graph. Another line of research [14, 22] models a hypergraph with a random hypergraph model, then clusters the hypergraph by iteratively maximizing the modularity-based objective scores.

Attributed hypergraph clustering (AHC) has also been studied. Existing AHC methods [10, 19, 32, 46] predominantly rely on matrix factorization techniques, leading to high computational and memory costs, thereby limiting scalability [32]. Additionally, they require prior knowledge of the number of clusters to produce quality clustering. However, the number of clusters for a desirable AHC is usually data-dependent and unknown beforehand, without knowing which, the performance can drop dramatically [65]. Specifically, JNMF [19] first represents an attributed hypergraph by a hypergraph Laplacian and an attribute matrix. It then integrates both by adopting a Non-negative Matrix Factorization (NMF) objective function that consists of an NMF part for the hypergraph topology and the attributes, respectively. [32] transforms an attributed hypergraph to an attributed dyadic graph and then extends GNMF [10], an NMF-based high-dimensional data clustering method, to three clustering algorithms GNMFA, GNMFC, GNMFL by clique reduction or hypergraph normalized Laplacian. However, the clique reduction hinders clustering effectiveness due to the information loss, and the NMF operations incur high computational and memory costs. To address the issue, GRAC [32] represents hypergraph topology by a less costly hypergraph Laplacian and then performs hypergraph convolution on node attributes to obtain a similarity matrix such that they better integrate the topological and attribute information. However, the following Singular Value Decomposition operations for clustering are still expensive, limiting its scalability.

Discussion on Similarity in Algorithm Flow with Existing Works. As summarized in Table 1, our AHRC shares three similar modules with existing works [28, 46]: an attribute graph module (AGM) for exploiting attribute information, a random walk module (RWM) for capturing topological information, and a clustering module (CLM). One of the major differences lies in how attribute and topology are integrated. Specifically, our AHRC first computes attribute similarities in AGM and topological similarities in RWM separately, and then integrates them in the representation integration module. In CLM, our method maximizes a modularity-based objective using the Louvain algorithm. In the solution of EDVW [28],

	Module		
	Attribute graph (AGM)	**Random walk (RWM)**	**Clustering (CLM)**
[28]	weighted incident matrix	topology	cut-based objective
[46]	KNN attribute graph	topology + attribute	cut-based objective
ours	KNN attribute graph	topology	modularity-based objective

Table 1: Similarity in Algorithm Flow with Existing Methods

all three modules, AGM, RWM, and CLM, are used. Specifcially, the AGM calculates tf-idf for each node-hyperedge pair to compute a weighted incident matrix. Then, the RWM constructs random-walk-based hypergraph Laplacians using the weighted incidence matrix. Note that, different from AHRC which computes the two similarities separately, EDVW computes them sequentially. Finally, the CLM adopts a cut-based objective function to obtain clustering via matrix factorization over the constructed Laplacians. Unlike our approach, EDVW carries out integration in the AGM module by directly computing the weighted incidence matrix. AHCKA [46] also uses AGM, RWM, and CLM. AGM first exploits attributes by using the KNN algorithm to compute an attribute graph. The RWM then integrates the attribute graph with hypergraph topology through a joint random walk process. Specifically, the random walk intertwines these two graphs: At each step, a walk can either move to a neighbor in the attribute graph or to a neighbor in the hypergraph, with a certain probability. However, it is unclear why the attribute similarity should and could be transmitted through random walk, especially when the attribute graph has considered attribute relations among all node pairs. Finally, the CLM adopts a cut-based objective function to obtain clustering by iteratively approximating the eigenvectors of the similarity matrix derived from the joint random walk. In contrast to our approach, AHCKA integrates the unweighted hypergraph topology and the attribute graph through a joint random walk in the RWM module.

We experimentally prove that our AHRC outperforms AHCKA [46] in the clustering quality; EDVW [28] primarily focuses on text datasets, and cannot be directly applied to the hypergraphs used in our experiments.

Measure the Similarity Among Nodes. The measures of node-wise topological [26, 55, 60, 65, 74] and attribute [30, 47, 70, 79] similarity in graphs have been extensively studied. These similarities can be stored in a similarity matrix, serving as an input for clustering algorithms such as Louvain and spectral method. A line of methods uses Gaussian similarity [65], L2 distance [74], or random walk [26, 55, 60] to compute node-wise similarities based on graph topology. However, these methods fail to capture the attribute information. Another line of methods [70, 79] augments the data graph by treating the attributes as 'nodes' and establishing a set of node-attribute associations. They then perform random walks on the augmented graph to integrate both topological and attribute information. However, these methods suffer from high computational costs on large graphs with multi-dimensional attributes. Additionally, considering all attributes with potential inconsistencies can diminish clustering effectiveness [46]. To capture sufficient attribute information while reducing computational cost and noise, methods [30, 47] employ the KNN algorithm to measure the attribute similarity.

Graph Contrastive Learning. Graph contrastive learning (GCL) methods [43, 68, 75, 81] learn an encoding function that takes node attributes and graph topology as input and produces node embeddings as output. This process involves using GNN layers to propagate and aggregate information based on the underlying graph structure. The embeddings can then be used for clustering by applying the k-Means algorithm. On dyadic graphs, the state-of-the-art GRACE [81] generates two graph views by randomly removing a portion of edges and node attributes, then learns node embeddings by maximizing the agreement that is measured by contrastive loss on node embeddings in these two views. However, GRACE cannot be directly applied to hypergraphs, necessitating the representation of hypergraphs into dyadic graphs through techniques such as clique reduction. There is still a lack of efficient and effective hypergraph representation methods. Among hypergraph contrastive learning methods [43, 68, 75], TRICL [43] achieves the state-of-the-art performances. It aggregates information directly on the underlying hypergraph structure, which can be represented by a hypergraph transition matrix capturing the local topology of 1-hop neighbors. However, as existing models are rather shallow – GRACE consists of two layers and TRICL has one layer only – they are inadequate to capture the global information. Under our AHR model, the AHR layer we proposed captures multi-hop relationships between nodes in both topological and attribute senses, effectively integrating global information to enhance the performance of contrastive learning on hypergraphs.

Name	**Dataset**	n	m	$\text{vol}(\mathcal{H})$	$\text{vol}_2(\mathcal{H})$	d
C13	C13-C [19]	693	545	3,475	26,908	4,728
WIK	Wiki [32]	1,999	2,184	16,321	91,479	4,973
COA	Cora-A [46]	2,708	1,072	4,585	17,136	1,433
COC	Cora-C [46]	2,708	1,579	4,786	5,687	1,433
CIC	Citeseer-C [46]	3,312	1,079	3,453	6,007	3,703
NEW	20News [46]	16,242	100	65,451	34,234,847	100
PBC	Pubmed-C [32]	19,717	7,963	34,629	186,155	500
DBA	DBLP-A [46]	41,302	22,363	99,561	906,564	1,425
AMZ	Amazon [53]	2,249,006	4,285,799	72,816,145	5,993,189,994	1,000
TWB	Tweibo [69]	2,320,895	50,133,382	100,266,764	61,186,099	1,657
MAG	MAGPM [46]	2,353,996	1,082,711	17,279,202	517,767,530	1,000

Table 2: Data Statistics

7 Experiments

This section evaluates the performance of our proposed AHRC method on 10 real-world attributed hypergraphs with ground truth clustering. To further demonstrate the scalability of our AHRC method, we include an additional attributed dyadic graph, *TWB*, with 2.3 million nodes and 50 million edges in the scalability test. Table 2 shows the data statistics. For the algorithmic AHC methods, all the experiments were conducted on a CPU server (Intel Xeon Gold 6230 CPU 2.10GHz, 376GB RAM, and Ubuntu 5.8.0-38-Generic). All methods were run 10 times to report the average. The cut-off running time was set to be 12 hours. For contrastive-learning-based AHC methods, all the experiments were conducted on a GPU server with an NVIDIA RTX A6000 48GB GPU. The implementation is available at anonymous GitHub repository[1].

Baselines. We compare our AHRC with 6 state-of-the-art algorithmic attributed hypergraph clustering methods that are introduced in Section 6: GNMFA, GNMFC, GNMFL [10], JNMF [19], GRAC [32], and AHCKA [46]. Additionally, we compare our contrastive learning methods TCL+ and GRC+ with the state-of-the-art

[1]https://anonymous.4open.science/r/Attributed_Hypergraph_Representation_for_Clustering_Code-4368

\	F-measure										ARI									
	C13	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*	*AMZ*	*MAG*	*C13*	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*	*AMZ*	*MAG*
GNMFA	0.13	0.35	0.23	0.22	0.19	\	0.13	0.42	\	\	0.01	0.24	0.01	0.00	0.02	\	0.00	0.20	\	\
GNMFC	0.12	0.35	0.26	0.24	0.29	\	**0.17**	0.41	\	\	0.01	0.24	0.07	0.03	0.15	\	0.04	0.19	\	\
GNMFL	0.14	0.30	0.28	0.27	0.42	\	**0.20**	0.49	\	\	0.02	0.17	0.14	0.07	0.28	\	0.03	0.37	\	\
JNMF	0.19	0.30	0.33	0.26	0.33	0.12	0.08	0.42	\	\	0.09	0.19	0.20	0.14	0.21	0.01	0.00	0.22	\	\
GRAC	**0.39**	0.32	0.38	0.33	0.28	0.14	0.14	0.56	\	\	**0.32**	0.20	0.27	0.22	0.11	0.07	0.05	0.45	\	\
AHCKA	0.38	**0.40**	**0.46**	**0.40**	**0.49**	**0.18**	**0.17**	**0.62**	**0.34**	**0.38**	0.31	**0.32**	**0.35**	**0.31**	**0.38**	**0.11**	**0.07**	**0.53**	**0.28**	**0.34**
AHRC	**0.40**	**0.41**	**0.55**	**0.51**	**0.46**	**0.28**	**0.17**	**0.62**	**0.61**	**0.43**	**0.33**	**0.34**	**0.46**	**0.41**	**0.36**	**0.20**	**0.06**	**0.53**	**0.54**	**0.37**

\	Jaccard Similarity										Purity									
	C13	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*	*AMZ*	*MAG*	*C13*	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*	*AMZ*	*MAG*
GNMFA	0.07	0.21	0.13	0.12	0.11	\	0.07	0.26	\	\	0.21	0.54	0.29	0.26	0.23	\	0.12	0.47	\	\
GNMFC	0.07	0.21	0.15	0.14	0.17	\	**0.09**	0.26	\	\	0.21	0.54	0.36	0.31	0.41	\	**0.20**	0.46	\	\
GNMFL	0.07	0.18	0.16	0.15	0.26	\	**0.11**	0.33	\	\	0.22	0.49	0.42	0.44	0.55	\	0.24	0.60	\	\
JNMF	0.10	0.17	0.20	0.15	0.19	0.06	0.04	0.27	\	\	0.25	0.47	0.50	0.38	0.47	0.18	0.06	0.52	\	\
GRAC	**0.24**	0.19	0.24	0.19	0.16	0.08	0.07	0.38	\	\	**0.39**	0.56	0.56	0.45	0.40	0.19	0.14	0.69	\	\
AHCKA	**0.24**	**0.25**	**0.30**	**0.25**	**0.33**	**0.10**	**0.09**	**0.45**	**0.21**	**0.23**	**0.39**	**0.54**	**0.61**	**0.55**	**0.64**	**0.22**	**0.19**	**0.74**	**0.71**	**0.63**
AHRC	**0.25**	**0.26**	**0.38**	**0.34**	**0.30**	**0.17**	**0.09**	**0.45**	**0.44**	**0.28**	**0.41**	**0.57**	**0.71**	**0.67**	**0.62**	**0.37**	**0.19**	**0.74**	**0.65**	**0.59**

\	Balanced Accuracy										NMI									
	C13	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*	*AMZ*	*MAG*	*C13*	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*	*AMZ*	*MAG*
GNMFA	0.51	**0.71**	0.51	0.50	0.51	\	0.50	0.66	\	\	0.12	0.46	0.05	0.02	0.02	\	0.00	0.30	\	\
GNMFC	0.51	**0.71**	0.54	0.52	0.57	\	0.52	0.65	\	\	0.12	0.46	0.11	0.05	0.18	\	0.15	0.28	\	\
GNMFL	0.53	0.68	0.56	0.54	0.65	\	0.51	0.69	\	\	0.16	0.46	0.20	0.21	0.27	\	0.09	0.43	\	\
JNMF	0.59	0.64	0.59	0.56	0.59	0.51	0.50	0.66	\	\	0.18	0.38	0.27	0.21	0.23	0.03	0.00	0.31	\	\
GRAC	**0.80**	0.68	0.62	0.59	0.56	0.53	0.52	0.73	\	\	**0.43**	0.46	0.36	0.34	0.22	0.19	**0.16**	0.53	\	\
AHCKA	**0.80**	0.69	**0.67**	**0.63**	**0.69**	**0.54**	**0.53**	**0.77**	**0.61**	**0.63**	0.42	**0.48**	**0.44**	**0.42**	**0.37**	**0.29**	**0.17**	**0.60**	**0.54**	**0.52**
AHRC	0.78	0.69	**0.73**	**0.69**	**0.67**	**0.58**	**0.53**	**0.77**	**0.77**	**0.67**	**0.44**	**0.52**	**0.49**	**0.46**	**0.36**	**0.27**	0.14	**0.61**	**0.47**	**0.49**

Table 3: Clustering Quality of Different Algorithmic Methods

\	F-measure								ARI							
	C13	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*	*C13*	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*
GRACE	0.14	0.36	0.26	0.31	0.28	\	**0.18**	\	0.01	0.28	0.05	0.16	0.07	\	**0.07**	\
GRC+	**0.29**	**0.40**	**0.35**	**0.31**	**0.35**	**0.19**	0.16	\	**0.21**	**0.33**	**0.22**	**0.19**	**0.21**	**0.11**	0.05	\
TRICL	0.17	0.21	0.44	0.37	0.45	0.19	0.16	0.56	0.04	0.06	0.34	0.26	0.35	0.12	**0.07**	0.44
TCL+	**0.39**	**0.39**	**0.46**	**0.43**	**0.46**	**0.23**	**0.20**	**0.65**	**0.31**	**0.31**	**0.37**	**0.32**	**0.36**	**0.14**	**0.07**	**0.57**

\	Jaccard Similarity								Purity							
	C13	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*	*C13*	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*
GRACE	0.07	0.22	0.15	**0.18**	0.16	\	**0.10**	\	0.26	0.52	0.40	0.47	0.39	\	**0.20**	\
GRC+	**0.17**	**0.25**	**0.21**	**0.18**	**0.21**	**0.11**	0.09	\	**0.35**	**0.56**	**0.55**	**0.48**	**0.50**	**0.26**	0.19	\
TRICL	0.09	0.12	0.28	0.23	0.29	0.11	0.08	0.39	0.28	0.35	0.61	0.54	0.59	0.22	0.17	0.71
TCL+	**0.24**	**0.24**	**0.30**	**0.27**	**0.30**	**0.13**	**0.11**	**0.49**	**0.38**	**0.55**	**0.62**	**0.59**	**0.61**	**0.27**	**0.24**	**0.77**

\	Balanced Accuracy								NMI							
	C13	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*	*C13*	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*
GRACE	0.52	0.67	0.53	0.58	0.55	\	**0.53**	\	0.22	0.51	0.17	**0.29**	0.16	\	**0.19**	\
GRC+	**0.68**	**0.68**	**0.61**	**0.59**	**0.61**	**0.54**	0.52	\	**0.34**	**0.54**	**0.31**	0.28	**0.29**	**0.24**	0.15	\
TRICL	0.61	0.58	0.65	0.62	**0.66**	0.54	**0.53**	0.75	0.22	0.29	0.43	0.40	0.38	0.32	0.19	0.58
TCL+	**0.79**	**0.68**	**0.66**	**0.65**	**0.66**	**0.55**	**0.53**	**0.78**	**0.44**	**0.47**	**0.44**	**0.45**	**0.39**	**0.33**	**0.21**	**0.64**

Table 4: Clustering Quality of Different Contrastive Learning Methods

contrastive learning methods TRICL [43] and GRACE [81]. For a fair comparison, we use the objective computed on node embeddings in TRICL.

Parameters. For all baselines, we adopt the default parameter values as suggested in their respective papers. Given that the number of clusters k in a desirable clustering is often not available in reality, our AHRC produces a reasonable k based on Property 1. For consistent comparison [22], this value of k will be used for baselines that require a predefined k. For our AHRC, unless otherwise specified, we set the default values of parameter $\alpha = 0.2$ following [46], $\tau = 3$, and $\gamma = 2$ based on our sensitivity analysis in Exp 5 and Exp 6.

Hyperparameters. We adopt a standard practice for hyperparameter tuning and model evaluation in graph contrastive learning [43, 61, 81]. We randomly divide graph nodes into a validation set (20%) and a test set (80%). We perform a grid search for hyperparameters (i.e., the learning rate, p_d and w_s in TCL+, and p_e and p_a in GRC+) based on validation clustering performance in F-measure, and choose the hyperparameters that yield the best performance. Then we use these hyperparameters on the test set. The selected hyperparameter values and detailed sensitivity tests are provided in the report [1].

7.1 Effectiveness and Scalability

In this section, we show the experimental results on clustering quality and scalability. Clustering quality is evaluated in the alignment to the ground truth clustering with 6 widely used metrics: F-measure [48], Adjusted Rand Index (ARI) [29], Jaccard Similarity [29], Purity [48], Balanced Accuracy [9], and Normalized Mutual Information (NMI) [35]. For all the above metrics, a larger score indicates better clustering quality.

Exp 1. Clustering Quality. Table 3 shows the clustering performance of our AHRC and 6 algorithmic baselines on datasets with ground truth. Top-2 scores for each dataset are highlighted with bold&underline and bold, respectively. '\' denotes no result due to time-out or out-of-memory reason. Baselines GNMFA, GNMFC, and GNMFL fail on dataset *NEW* as they regard the entire graph

\	*C13*		*WIK*		*COA*		*COC*		*CIC*		*NEW*		*PBC*		*DBA*		*AMZ*		*MAG*	
	Time	Space	Time	Space	Time	Space	Time	Space	Time	Space	Time	Space	Time	Space	Time	Space	Time	Space	Time	Space
GNMFA	23.69	0.15	112.74	0.30	31.31	0.27	27.30	0.27	164.84	0.35	295.24	2.21	414.93	4.29	1,550.46	20.17	\	\	\	\
GNMFC	23.90	0.15	118.46	0.27	32.54	0.23	27.30	0.23	163.70	0.31	301.69	2.14	411.21	4.27	1,501.95	20.17	\	\	\	\
GNMFL	23.81	0.15	121.23	0.21	29.55	0.17	28.92	0.17	168.01	0.17	315.50	2.13	483.23	4.34	2,381.92	20.11	\	\	\	\
JNMF	1.58	0.15	6.40	0.31	10.73	0.21	11.04	0.24	30.19	0.29	284.69	2.15	493.89	4.29	2,716.49	20.11	\	\	\	\
GRAC	35.15	0.23	89.51	0.39	32.97	0.36	14.17	0.36	61.27	0.41	**30.60**	0.29	**19.29**	0.43	525.27	1.32	\	\	\	\
AHCKA	**0.74**	**0.03**	**2.93**	**0.14**	**1.98**	**0.02**	**3.47**	**0.13**	**2.67**	**0.04**	42.48	**0.13**	43.86	**0.19**	**8.74**	**0.21**	**18,799.85**	**6.16**	**11,064.37**	**5.34**
AHRC	**0.16**	**0.03**	**0.47**	**0.04**	**0.48**	**0.02**	**0.74**	**0.02**	**0.47**	**0.02**	**33.37**	**0.16**	**3.70**	**0.07**	**9.53**	**0.19**	**6,213.66**	**21.80**	**4,235.63**	**16.19**

Table 5: Time and Memory Cost of Different Algorithmic Methods (Time in Seconds, RAM in GBs)

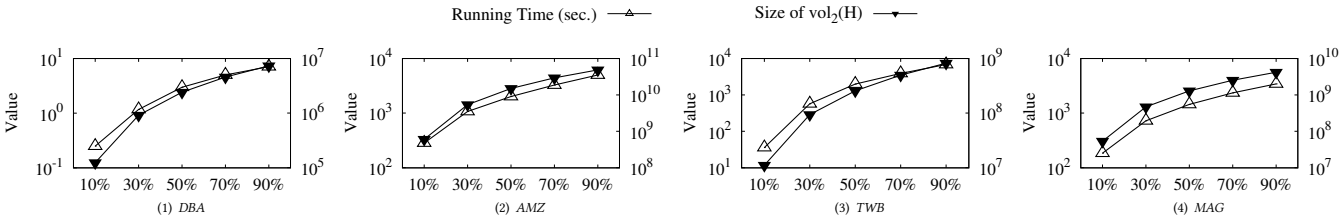


Figure 5: Scalability: Time Cost of AHRC by Varying Node Number

as a single cluster. In general, our AHRC achieves the best overall performance. Specifically, on F-measure, AHRC surpasses all 6 baselines (in top-down order as listed in Table 3 unless otherwise specified) by 102%, 83%, 61%, 80%, 41%, and 20%, respectively, averaged over all datasets. In terms of ARI, AHRC outperforms all 6 baselines by 8, 329%, 776%, 598%, 6, 585%, 85%, and 24%, respectively. On Jaccard Similarity, AHRC is 133%, 104%, 86%, 107%, 53%, and 26% higher than the 6 baselines. For Purity, AHRC outperforms the baselines by 98%, 60%, 34%, 75%, 34%, and 10%, respectively. On Balanced Accuracy, AHRC outperforms baselines by 26%, 22%, 18%, 17%, 9%, and 5%, respectively. On NMI, AHRC is 4, 645%, 237%, 83%, 2, 370%, and 24% higher than GNMFA, GNMFC, GNMFL, JNMF, and GRAC, respectively. AHRC obtains a similar (by an average of −1% lower) NMI to AHCKA.

Exp 2. Clustering Quality. Table 4 shows the clustering performance of our TCL+, GRC+, and 2 contrastive learning baselines on datasets with ground truth. Due to the out-of-memory reason, none of the methods could run on the datasets with millions of nodes such as *MAG*. In general, our TCL+ achieves the best overall performance among all 4 methods. Averaged across all datasets, TCL+ obtains 694%, 45%, and 147% higher ARI than GRACE, GRC+, and TRICL, respectively, and outperforms them by 78%, 33%, and 25% in terms of NMI, respectively.

To better demonstrate the effectiveness of our models and AHR layer, we group TCL+ with TRICL, and GRC+ with GRACE, highlighting the best score within each group for each dataset. Specifically, our TCL+ constantly outperforms TRICL on all datasets across all 6 metrics. Specifically, TCL+ achieves 38% higher F-measure, 147% higher ARI, 47% higher Jaccard Similarity, 22% higher Purity, 7% higher Balanced Accuracy, and 25% higher NMI compared to TRICL, averaged over all datasets. Comparing our GRC+ and GRACE, GRC+ constantly surpasses GRACE on all datasets except *PBC* across all 6 metrics: averaged over all datasets, GRC+ achieves 28%, 425%, 36%, 18%, 10%, and 33% higher F-measure, ARI, Jaccard Similarity, Purity, Balanced Accuracy, and NMI, respectively, than GRACE.

Exp 3. Time and Memory Cost. Table 5 shows the time and memory cost of AHRC and the algorithmic baselines on 10 hypergraphs. On the largest hypergraphs *AMZ* and *MAG*, the baselines GRAC failed with out-of-time errors, and the other baselines failed with out-of-memory errors. In terms of running time, AHRC is in general the fastest among all 7 methods, showing the effectiveness of our method. Specifically, over all datasets, GNMFA, GNMFC, GNMFL, JNMF, GRAC, and AHCKA is on average 140, 141, 156, 68, 85, and 4 times slower than our AHRC, respectively. In terms of memory cost, averaged over all datasets, GNMFA, GNMFC, GNMFL, JNMF, and GRACE take 29, 28, 26, 28, and 10 times more memory space than our AHRC, respectively. On *AMZ* and *MAG*, we take 2.5 and 2 times more memory space than AHRC, respectively. It is because we store the transition matrix $\mathbf{T}$ of a hypergraph for spanning forest sparsification, while AHCKA maintains 3 small matrices: i) a matrix of size $O((k+1)n)$ to approximate the top $k+1$ eigenvectors (k is the number of clusters) of $\mathbf{T}$ for clustering, ii) the $O(\text{vol}(\mathcal{H}))$ incident matrix, and iii) the $O(Kn)$ transition matrix of the KNN attribute grap (K is the number of neighbors in the attribute graph). However, the running time of AHCKA is much higher than that of AHRC for the iterative matrix multiplications. Specifically, AHCKA iteratively updates the approximations by recomputing matrix multiplications on these matrices in each iteration.

Exp 4. Scalability. Figure 5 shows the time cost of AHRC (on the left y-axis) and the dyadic volume of the graphs (on the right y-axis) when varying the number of nodes of the graph. Due to the space limit, we only show the results on 4 largest datasets. For each graph, we created 5 induced subgraphs, containing 10%, 30%, 50%, 70%, and 90% of nodes randomly selected from the original graph, respectively, and reported the running time on them. Figures 5 shows that the time cost exhibits approximately a linear trend with the dyadic volume $\text{vol}_2(\mathcal{H})$. This result echoes the complexity of AHRC proved by Lemma 5. It also underscores the primary scalability limitation discussed in Section 4, where the growing $\text{vol}_2(\mathcal{H})$ impacts scalability.

7.2 Sensitivity

Exp 5. Sensitivity Test on τ. We conduct a sensitivity test by varying the parameter τ from 1 to 6 to test its impact on the clustering quality and time cost of AHRC. Figure 6 reports the scores for 6 metrics on 8 datasets. As a general trend, increasing τ slightly improves the clustering quality. With a small τ, the sparsified graph captures insufficient topological relationships, which degrades the clustering quality. As τ increases, the clustering quality improves,

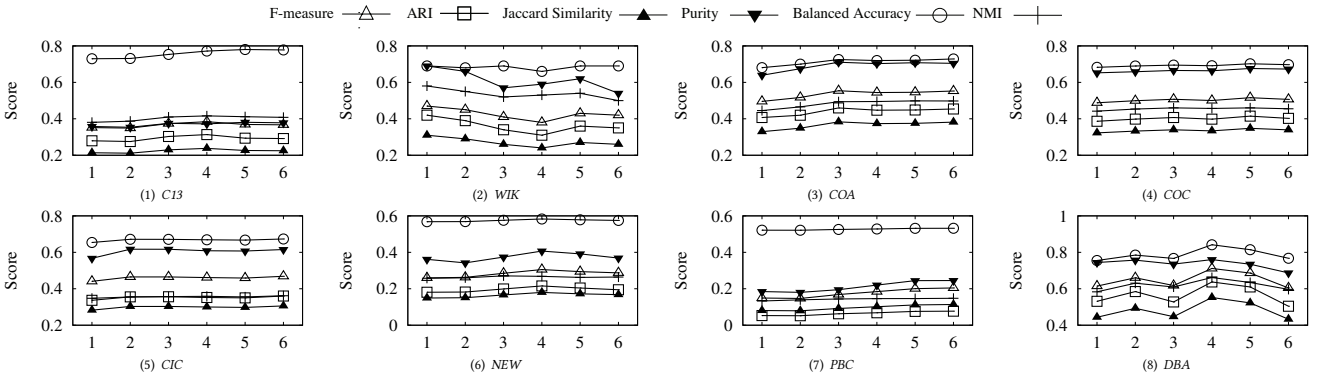


Figure 6: Sensitivity: Clustering Quality of AHRC by Varying τ

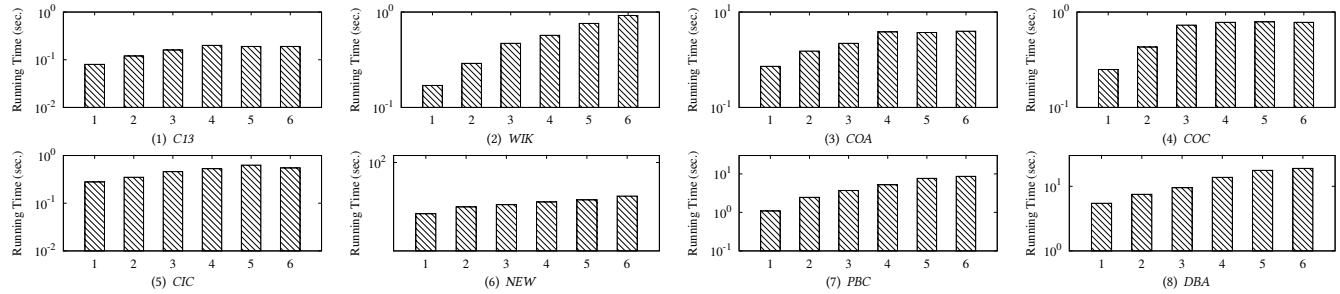


Figure 7: Sensitivity: Time Cost of AHRC by Varying τ

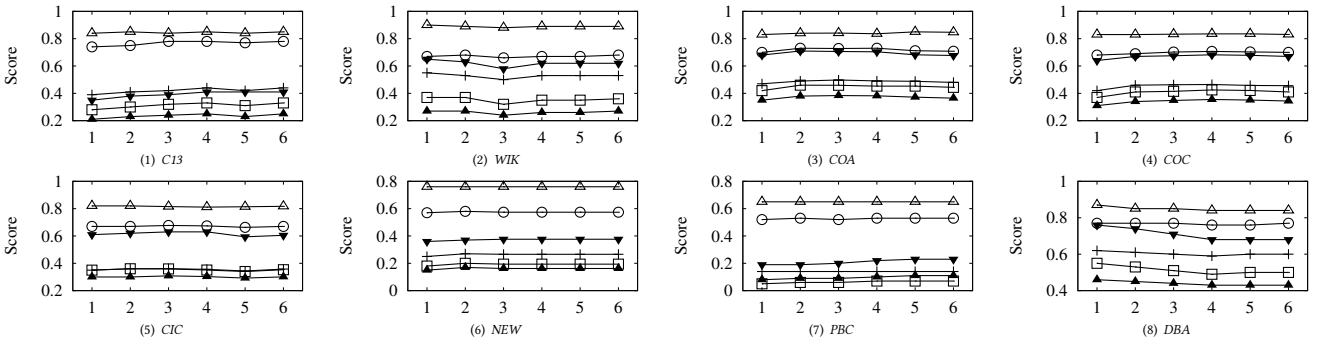


Figure 8: Sensitivity: Clustering Quality of AHRC by Varying γ

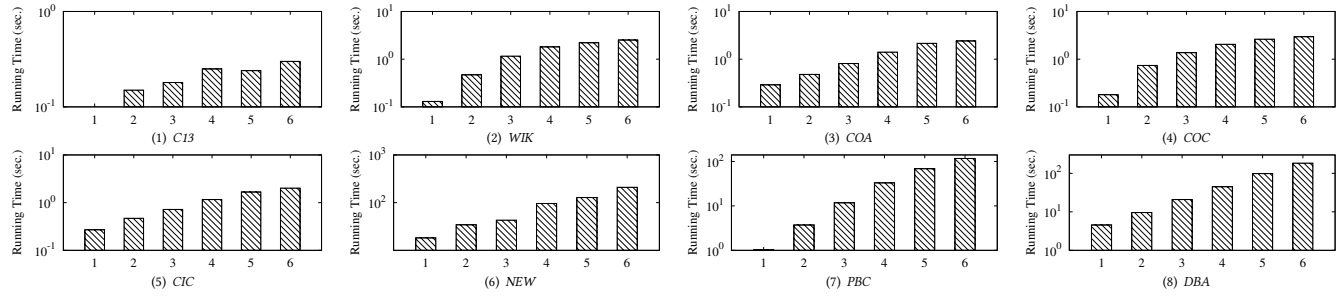


Figure 9: Sensitivity: Time Cost of AHRC by Varying γ

as more topological structure is preserved. When τ becomes greater than 4, the quality may slightly degrade on some datasets (e.g., *NEW* and *DBA*). Overall, the clustering quality exhibits minor sensitivity to the change in τ. As τ increases from 1 to 6, the metric scores change by 4.5%, −0.6%, 2.4%, −0.7%, and 0.7% at each step, respectively, averaged on all metrics and datasets. Figure 7 shows the time cost of AHRC. In general, the running time of AHRC rises as τ increases. Specifically, when τ increases from 2 to 6, AHRC takes 53%, 107%, 158%, 205%, and 221% more time compared to $\tau = 1$. This result echoes the complexity proved by Lemma 5. To achieve a good balance between the clustering effectiveness and efficiency, we set $\tau = 3$ as the default value.

Exp 6. Sensitivity Test on γ. We evaluate the clustering quality and time cost of AHRC when the parameter γ varies from 1 to 6.

\	*C13*		*WIK*		*COA*		*COC*		*CIC*		*NEW*		*PBC*		*DBA*		*AMZ*		*MAG*	
	F1	Time	F1	Time	F1	Time	F1	Time	F1	Time	F1	Time	F1	Time	F1	Time	F1	Time	F1	Time
KN	**0.37**	**0.34**	0.42	**0.86**	**0.54**	**0.57**	**0.49**	**0.65**	**0.46**	**1.04**	**0.28**	**21.92**	0.15	**5.12**	**0.64**	**21.86**	0.26	6,705.02	**0.40**	6934.25
ER	0.35	5.66	0.43	18.88	**0.54**	12.90	**0.49**	19.95	**0.47**	20.66	0.26	10862.59	0.29	466.01	0.60	2368.47	\	\	\	\
ERW	0.33	3.82	0.44	13.41	0.53	0.83	**0.49**	16.68	**0.47**	15.88	0.25	7703.68	**0.27**	405.41	**0.63**	1755.54	\	\	\	\
LSim	0.24	1.93	**0.47**	7.40	0.51	1.37	0.48	1.27	**0.46**	2.38	\	\	0.21	50.73	0.61	82.95	\	\	\	\
LS	0.27	1.55	**0.45**	6.14	0.51	1.15	0.48	1.10	**0.46**	1.74	\	\	0.22	44.18	0.62	66.34	\	\	\	\
LD	0.35	0.38	0.42	1.23	0.51	1.87	**0.49**	1.67	**0.46**	1.67	**0.32**	**17.71**	**0.31**	32.91	0.62	29.98	**0.50**	**3305.32**	0.37	**5098.33**
SPF	**0.37**	**0.11**	0.41	**0.32**	**0.55**	**0.54**	**0.51**	**0.34**	**0.46**	**0.34**	**0.28**	23.75	0.17	**2.72**	0.62	**6.12**	**0.60**	**6,213.66**	**0.43**	**4235.63**

Table 6: Sensitivity: Clustering Quality and Time Cost of AHRC with Different Sparsification Methods

\	F-measure										ARI									
	C13	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*	*AMZ*	*MAG*	*C13*	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*	*AMZ*	*MAG*
AHRC (LIN)	0.36	0.46	0.45	0.43	0.38	0.22	0.13	0.52	0.16	0.21	0.29	0.41	0.36	0.34	0.28	0.15	0.05	0.43	0.12	0.15
AHRC (LOG)	0.36	0.45	0.34	0.34	0.33	0.22	0.13	0.48	0.14	0.15	0.30	0.41	0.24	0.25	0.24	0.15	0.04	0.37	0.11	0.10
AHRC (EXP)	0.24	**0.47**	0.43	0.39	0.38	0.22	0.13	0.50	0.16	0.21	0.17	**0.42**	0.34	0.29	0.28	0.15	0.05	0.40	0.13	0.15
AHRC (SQR)	**0.40**	0.41	**0.55**	**0.51**	**0.46**	**0.28**	**0.17**	**0.62**	**0.61**	**0.43**	**0.33**	0.34	**0.46**	**0.41**	**0.36**	**0.20**	**0.06**	**0.53**	**0.54**	**0.37**

Table 7: Sensitivity: Clustering Quality of AHRC with Different Transformation Functions

\	F-measure									
	C13	*WIK*	*COA*	*COC*	*CIC*	*NEW*	*PBC*	*DBA*	*AMZ*	*MAG*
AHRC \ ATM	0.14	0.16	0.21	0.11	0.10	0.23	0.02	0.27	0.22	0.34
AHRC \ RWM	**0.38**	0.34	**0.47**	**0.48**	**0.40**	0.19	**0.23**	**0.59**	**0.26**	**0.38**
AHRC \ $\mathbf{S_T}$	0.21	**0.40**	0.29	0.29	0.29	**0.24**	0.14	0.45	0.12	0.12
AHRC \ $\mathbf{S_A}$	0.16	0.13	0.05	0.06	0.01	0.01	0.02	0.06	0.01	0.01
AHRC	**0.37**	**0.41**	**0.55**	**0.51**	**0.46**	**0.28**	**0.17**	**0.62**	**0.60**	**0.43**

Table 8: Impact of Individual Modules on Clustering Quality

Figure 8 reports the scores for 6 metrics on 8 datasets. As a general trend, the clustering quality increases sharply when γ increase from 1 to 2 and then remains relatively stable as γ grows larger. Specifically, as γ increases from 1 to 6, the metric scores change by 7.9%, −1%, 2.2%, −0.8%, and 0.8% at each step, respectively, averaged on all metrics and datasets. Figure 9 shows the time cost of AHRC. In general, the running time of AHRC rises as γ increases. Specifically, when γ increases from 2 to 6, AHRC takes 76%, 224%, 470%, and 789% more time compared to $\gamma = 1$. This result echoes Lemma 1. Thus, considering the efficiency, we set $\gamma = 2$ as the default value.

Exp 7. Sparsification Methods. We conduct a sensitivity test to evaluate how different sparsification methods impact the clustering quality of AHRC. In addition to spanning forest sparsification (SPF), we tested 6 top-performing deterministic sparsification methods from [13]: K-Neighbor (KN), ER-unweighted (ER), ER-weighted (ERW), Local Similarity (LSim), LSpar (LS), and Local Degree (LD). For a fair comparison, all methods were set to the same sparsification level, with each node having 6 neighbors. Table 6 reports the F-measure score and time cost on 10 datasets. SPF in general achieves the fastest performance. Specifically, KN, ER, ERW, LSim, LS, and LD are on average 40%, 98%, 91%, 85%, 82%, and 53% slower than SPF, respectively, over all datasets. In terms of the clustering quality, SPF outperforms KN, LSim and LS by 15%, 5% and 3%, and outperforms slightly worse than ER, ERW and LD by −3%, −2% and −1%. Note that on the largest datasets *AMZ* and *MAG*, ER, ERW, LSim, and LS exceeded the time limit due to their low efficiency. Among the rest, our SPF achieves the best clustering quality. Overall, SPF provides the best trade-off between the effectiveness and efficiency, making it the most suitable choice for AHRC pipeline.

Exp 8. Transformation Functions. We evaluate the clustering performance of AHRC using the square root transformation compared to other transformation methods. We compare four variants of our method using 4 transformation functions ρ: AHRC (LIN) denotes our algorithm using $\rho(s) = s$, AHRC (LOG) uses $\rho(s) = \frac{s}{\log(1/s+1)}$, AHRC (EXP) uses $\rho(s) = \frac{\exp(s)-1}{\exp(1)-1}$, where $\exp(1) = 2.71828\cdots$, and AHRC (SQR) uses $\rho(s) = \sqrt{s}$. Table 7 reports the F-measure and ARI scores. In general, AHRC (SQR) achieves the best clustering performance. Specifically, on F-measure, AHRC (SQR) outperforms the other 3 variants (in top-down order as listed in Table 7) by 22%, 40%, and 29%, respectively, averaged over all datasets. On ARI, it outperforms the other 3 variants by 27%, 56%, and 38%, respectively.

Exp 9. Ablation Studies. To quantify the impact of different modules, we conduct ablation studies by creating 4 variants of our method. For the attribute graph module, AHRC\ATM replaces the KNN graph with a KNN adjacency graph, where each node selects its k most similar neighbors based on attribute similarity among its adjacent nodes. For the random walk module, AHRC\RWM skips random walk. For the integration module, AHRC\TSM uses only $\mathbf{S_A}$ and AHRC \ ASM uses only $\mathbf{S_T}$. Table 8 shows the F-measure score on 10 datasets. Overall, AHRC achieves the best performance. Specifically AHRC achieves 231%, 23%, 104%, and 2108% higher F-measure scores than the other variants (in top-down order as listed), respectively, averaged over all datasets. The ablation studies show the impact of each module on the clustering performance.

8 Conclusions

This paper proposes AHRC, an attributed hypergraph clustering approach with cutting-edge clustering quality and scalability. The performance of AHRC attributes to three new designs of AHRC compared to existing methods: 1) a novel integration of multi-hop hypergraph topology and attributed information, 2) a new formulation multi-hop modularity for clustering, 3) an effective sparsification for improving the scalability, and 4) generalizability to enhance contrastive learning. Our experiments show that AHRC significantly outperforms the state-of-the-art methods on real-world hypergraphs and in particular, it is up to two orders of magnitude faster than the baseline methods.

References

[1] Sameer Agarwal, Jongwoo Lim, Lihi Zelnik-Manor, Pietro Perona, David J. Kriegman, and Serge J. Belongie. 2005. Beyond Pairwise Clustering. In *CVPR*. 838–845. https://doi.org/10.1109/CVPR.2005.89

[2] William Aiello, Fan Chung, and Linyuan Lu. 2000. A random graph model for massive graphs. In *STOC*. 171–180. https://doi.org/10.1145/335305.335326

[3] Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon M. Kleinberg. 2018. Simplicial closure and higher-order link prediction. *Proc. Natl.*

Acad. Sci. USA 115, 48 (2018), E11221–E11230. https://doi.org/10.1073/PNAS.1800683115

[4] Aritra Bhowmick, Mert Kosan, Zexi Huang, Ambuj K. Singh, and Sourav Medya. 2024. DGCLUSTER: A Neural Framework for Attributed Graph Clustering via Modularity Maximization. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*. AAAI Press, 11069–11077. https://doi.org/10.1609/AAAI.V38I10.28983

[5] Vincent Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast Unfolding of Communities in Large Networks. *J. Stat. Mech.* 2008, 10 (04 2008), P10008. https://doi.org/10.1088/1742-5468/2008/10/P10008

[6] Béla Bollobás and Bela Bollobas. 1998. *Modern graph theory*. Vol. 184. Springer Science & Business Media. https://doi.org/10.1007/978-1-4612-0619-4

[7] Cécile Bothorel, Juan David Cruz, Matteo Magnani, and Barbora Micenková. 2015. Clustering attributed graphs: Models, measures and methods. *Netw. Sci.* 3, 3 (2015), 408–444. https://doi.org/10.1017/NWS.2015.9

[8] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. 2006. Maximizing modularity is hard. *arXiv preprint physics/0608255* (2006).

[9] Kay Henning Brodersen, Cheng Soon Ong, Klaas Enno Stephan, and Joachim M. Buhmann. 2010. The Balanced Accuracy and Its Posterior Distribution. In *ICPR*. 3121–3124. https://doi.org/10.1109/ICPR.2010.764

[10] Deng Cai, Xiaofei He, Jiawei Han, and Thomas S. Huang. 2011. Graph Regularized Nonnegative Matrix Factorization for Data Representation. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 8 (2011), 1548–1560. https://doi.org/10.1109/TPAMI.2010.231

[11] Jie Chen, Haw-ren Fang, and Yousef Saad. 2009. Fast Approximate *k*NN Graph Construction for High Dimensional Data via Recursive Lanczos Bisection. *J. Mach. Learn. Res.* 10 (2009), 1989–2012. https://doi.org/10.5555/1577069.1755852

[12] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 1597–1607.

[13] Yuhan Chen, Haojie Ye, Sanketh Vedula, Alex M. Bronstein, Ronald G. Dreslinski, Trevor N. Mudge, and Nishil Talati. 2023. Demystifying Graph Sparsification Algorithms in Graph Properties Preservation. *Proc. VLDB Endow.* 17, 3 (2023), 427–440. https://www.vldb.org/pvldb/vol17/p427-chen.pdf

[14] Philip S. Chodrow, Nate Veldt, and Austin R. Benson. 2021. Generative hypergraph clustering: From blockmodels to modularity. *Science Advances* 7, 28 (2021), eabh1303. https://doi.org/10.1126/sciadv.abh1303

[15] Fan Chung and Linyuan Lu. 2002. The average distances in random graphs with given expected degrees. *Natl Acad. Sci.* 99, 25 (2002), 15879–15882. https://doi.org/10.1073/pnas.252631999

[16] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Phys. Rev. E* 70, 6 (2004), 066111. https://doi.org/10.1103/PhysRevE.70.066111

[17] David Combe, Christine Largeron, Elöd Egyed-Zsigmond, and Mathias Géry. 2012. Combining Relations and Text in Scientific Network Clustering. In *International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2012, Istanbul, Turkey, 26-29 August 2012*. IEEE Computer Society, 1248–1253. https://doi.org/10.1109/ASONAM.2012.215

[18] Robin Devooght, Amin Mantrach, Ilkka Kivimäki, Hugues Bersini, Alejandro Jaimes, and Marco Saerens. 2014. Random walks based modularity: application to semi-supervised learning. In *23rd International World Wide Web Conference, WWW '14*. 213–224. https://doi.org/10.1145/2566486.2567986

[19] Rundong Du, Barry L. Drake, and Haesun Park. 2019. Hybrid clustering based on content and connection structure using joint nonnegative matrix factorization. *J. Glob. Optim.* 74, 4 (2019), 861–877. https://doi.org/10.1007/s10898-017-0578-x

[20] Jordi Duch and Alex Arenas. 2005. Community detection in complex networks using extremal optimization. *Phys. Rev. E* 72, 2 (2005), 027104. https://doi.org/10.1103/PhysRevE.72.027104

[21] Issam Falih, Nistor Grozavu, Rushed Kanawati, and Younès Bennani. 2018. Community detection in Attributed Network. In *WWW*. ACM, 1299–1306. https://doi.org/10.1145/3184558.3191570

[22] Zijin Feng, Miao Qiao, and Hong Cheng. 2023. Modularity-based Hypergraph Clustering: Random Hypergraph Model, Hyperedge-cluster Relation, and Computation. *Proc. ACM Manag. Data* 1, 3, Article 215 (nov 2023), 25 pages. https://doi.org/10.1145/3617335

[23] Santo Fortunato. 2010. Community detection in graphs. *Phys. Reps.* 486, 3-5 (2010), 75–174. https://doi.org/10.1016/j.physrep.2009.11.002

[24] Andrew Gelman and Jennifer Hill. 2007. *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press.

[25] Rumi Ghosh and Kristina Lerman. 2008. Community Detection Using a Measure of Global Influence. In *Advances in Social Network Mining and Analysis, Second International Workshop, SNAKDD 2008 (Lecture Notes in Computer Science, Vol. 5498)*. 20–35. https://doi.org/10.1007/978-3-642-14929-0_2

[26] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *SIGKDD*. 855–864. https://doi.org/10.1145/2939672.2939754

[27] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*. PMLR, 3887–3896.

[28] Koby Hayashi, Sinan G. Aksoy, Cheong Hee Park, and Haesun Park. 2020. Hypergraph Random Walks, Laplacians, and Clustering. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*. ACM, 495–504. https://doi.org/10.1145/3340531.3412034

[29] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *J. Classif.* 2, 1 (1985), 193–218. https://doi.org/10.1007/BF01908075

[30] Caiyan Jia, Yafang Li, Matthew B Carson, Xiaoyang Wang, and Jian Yu. 2017. Node attribute-enhanced community detection in complex networks. *Scientific reports* 7, 1 (2017), 2626.

[31] Jinhong Jung, Namyong Park, Lee Sael, and U Kang. 2017. BePI: Fast and Memory-Efficient Method for Billion-Scale Random Walk with Restart. In *SIGMOD*. ACM, 789–804. https://doi.org/10.1145/3035918.3035950

[32] Barakeel Fanseu Kamhoua, Lin Zhang, Kaili Ma, James Cheng, Bo Li, and Bo Han. 2021. HyperGraph Convolution Based Attributed HyperGraph Clustering. In *CIKM*. ACM, 453–463. https://doi.org/10.1145/3459637.3482437

[33] Bogumił Kamiński, Valérie Poulin, Paweł Prałat, Przemysław Szufel, and François Théberge. 2019. Clustering via hypergraph modularity. *PloS one* 14, 11 (2019), e0224307. https://doi.org/10.1371/journal.pone.0224307

[34] Ravi Kannan, Santosh S. Vempala, and Adrian Vetta. 2004. On clusterings: Good, bad and spectral. *J. ACM* 51, 3 (2004), 497–515. https://doi.org/10.1145/990308.990313

[35] Min-Soo Kim and Jiawei Han. 2009. A particle-and-density based evolutionary clustering method for dynamic networks. *VLDB* 2, 1 (2009), 622–633. https://doi.org/10.14778/1687627.1687698

[36] Sungwoong Kim, Sebastian Nowozin, Pushmeet Kohli, and Chang Yoo. 2011. Higher-order correlation clustering for image segmentation. *NIPS* 24 (2011), 1530–1538.

[37] Jon M. Kleinberg and Éva Tardos. 2006. *Algorithm design*. Addison-Wesley.

[38] Larkshmi Krishnamurthy, Joseph Nadeau, Gultekin Ozsoyoglu, M Ozsoyoglu, Greg Schaeffer, Murat Tasan, and Wanhong Xu. 2003. Pathways database system: an integrated system for biological pathways. *Bioinformatics* 19, 8 (2003), 930–937. https://doi.org/10.1093/bioinformatics/btg113

[39] Gayan K. Kulatilleke, Marius Portmann, and Shekhar S. Chandra. 2022. SCGC : Self-Supervised Contrastive Graph Clustering. *CoRR* abs/2204.12656 (2022). https://doi.org/10.48550/ARXIV.2204.12656 arXiv:2204.12656

[40] Tarun Kumar, Sankaran Vaidyanathan, Harini Ananthapadmanabhan, Srinivasan Parthasarathy, and Balaraman Ravindran. 2020. Hypergraph clustering by iteratively reweighted modularity maximization. *Appl. Netw. Sci.* 5, 1 (2020), 52. https://doi.org/10.1007/S41109-020-00300-3

[41] Tarun Kumar, Sankaran Vaidyanathan, Harini Ananthapadmanabhan, Srinivasan Parthasarathy, and Balaraman Ravindran. 2020. Hypergraph clustering by iteratively reweighted modularity maximization. *Appl. Netw. Sci.* 5, 1 (2020), 52. https://doi.org/10.1007/s41109-020-00300-3

[42] Kevin J. Lang. 2005. Fixing two weaknesses of the Spectral Method. In *NIPS*. 715–722.

[43] Dongjin Lee and Kijung Shin. 2023. I'm Me, We're Us, and I'm Us: Tri-directional Contrastive Learning on Hypergraphs. In *AAAI*. AAAI Press, 8456–8464. https://doi.org/10.1609/AAAI.V37I7.26019

[44] Pan Li and Olgica Milenkovic. 2017. Inhomogeneous Hypergraph Clustering with Applications. In *NIPS*, Vol. 30. 2308–2318.

[45] Pan Li and Olgica Milenkovic. 2018. Submodular hypergraphs: p-laplacians, cheeger inequalities and spectral clustering. In *ICML*. 3014–3023.

[46] Yiran Li, Renchi Yang, and Jieming Shi. 2023. Efficient and Effective Attributed Hypergraph Clustering via K-Nearest Neighbor Augmentation. *Proc. ACM Manag. Data* (2023), 116:1–116:23. https://doi.org/10.1145/3589261

[47] Changshu Liu, Liangjian Wen, Zhao Kang, Guangchun Luo, and Ling Tian. 2021. Self-supervised consensus representation learning for attributed graph. In *Proceedings of the 29th ACM international conference on multimedia*. 2654–2662.

[48] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*. Cambridge University Press. https://doi.org/10.1017/CBO9780511809071

[49] Boaz Nadler and Meirav Galun. 2006. Fundamental Limitations of Spectral Clustering. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, Bernhard Schölkopf, John C. Platt, and Thomas Hofmann (Eds.). MIT Press, 1017–1024.

[50] Jennifer Neville, Micah Adler, and David Jensen. 2003. Clustering relational data using attribute and link information. In *Proceedings of the text mining and link analysis workshop, 18th international joint conference on artificial intelligence*. San Francisco, CA: Morgan Kaufmann Publishers, 9–15.

[51] Mark Newman. 2010. *Networks: An Introduction*. Oxford University Press. https://doi.org/10.1093/acprof:oso/9780199206650.001.0001

[52] Mark EJ Newman. 2006. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E* 74, 3 (2006), 036104. https://doi.org/10.1103/PhysRevE.74.036104

[53] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*. 188–197.

[54] RL Ott and Michael Longnecker. 2016. *An introduction to statistical methods and data analysis*. Cengage Learning.

[55] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *SIGKDD*. 701–710. https://doi.org/10.1145/2623330.2623732

[56] Satu Elisa Schaeffer. 2007. Graph clustering. *Comput. Sci. Rev.* 1, 1 (2007), 27–64. https://doi.org/10.1016/j.cosrev.2007.05.001

[57] Jianbo Shi and J. Malik. 2000. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 8 (2000), 888–905. https://doi.org/10.1109/34.868688

[58] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *TPAMI* 22, 8 (2000), 888–905. https://doi.org/10.1109/34.868688

[59] Yuuki Takai, Atsushi Miyauchi, Masahiro Ikeda, and Yuichi Yoshida. 2020. Hypergraph Clustering Based on PageRank. In *KDD*. 1970–1978. https://doi.org/10.1145/3394486.3403248

[60] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *WWW*. 1067–1077. https://doi.org/10.1145/2736277.2741093

[61] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L. Dyer, Rémi Munos, Petar Velickovic, and Michal Valko. 2022. Large-Scale Representation Learning on Graphs via Bootstrapping. In *ICLR*. OpenReview.net. https://openreview.net/forum?id=0UXT6PpRpW

[62] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast Random Walk with Restart and Its Applications. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*. IEEE Computer Society, 613–622. https://doi.org/10.1109/ICDM.2006.70

[63] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2018. Deep graph infomax. *arXiv preprint arXiv:1809.10341* (2018).

[64] Ulrike von Luxburg. 2007. A tutorial on spectral clustering. *Stat. Comput.* 17, 4 (2007), 395–416. https://doi.org/10.1007/S11222-007-9033-Z

[65] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and computing* 17 (2007), 395–416. https://doi.org/10.1007/s11222-007-9033-z

[66] Joyce Jiyoung Whang, Rundong Du, Sangwon Jung, Geon Lee, Barry L. Drake, Qingqing Liu, Seonggoo Kang, and Haesun Park. 2020. MEGA: Multi-View Semi-Supervised Clustering of Hypergraphs. *Proc. VLDB Endow.* 13, 5 (2020), 698–711. https://doi.org/10.14778/3377369.3377378

[67] Ming-Juan Wu, Ying-Lian Gao, Jin-Xing Liu, Chun-Hou Zheng, and Juan Wang. 2019. Integrative hypergraph regularization principal component analysis for sample clustering and co-expression genes network analysis on multi-omics data. *IEEE Journal of Biomedical and Health Informatics* 24, 6 (2019), 1823–1834.

[68] Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Lizhen Cui, and Xiangliang Zhang. 2021. Self-Supervised Hypergraph Convolutional Networks for Session-based Recommendation. In *AAAI*. AAAI Press, 4503–4511. https://doi.org/10.1609/AAAI.V35I5.16578

[69] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S. Bhowmick. 2020. Homogeneous Network Embedding for Massive Graphs via Reweighted Personalized PageRank. *Proc. VLDB Endow.* 13, 5 (2020), 670–683. https://doi.org/10.14778/3377369.3377376

[70] Renchi Yang, Jieming Shi, Yin Yang, Keke Huang, Shiqi Zhang, and Xiaokui Xiao. 2021. Effective and Scalable Clustering on Massive Attributed Graphs. In *WWW*. ACM / IW3C2, 3675–3687. https://doi.org/10.1145/3442381.3449875

[71] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. 4805–4815. https://proceedings.neurips.cc/paper/2018/hash/e77dbaf6759253c7c6d0efc5690369c7-Abstract.html

[72] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph Contrastive Learning with Augmentations. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. https://proceedings.neurips.cc/paper/2020/hash/3fe230348e9a12c13120749e3f9fa4cd-Abstract.html

[73] Raphael Yuster and Uri Zwick. 2005. Fast sparse matrix multiplication. *ACM Trans. Algorithms* 1, 1 (2005), 2–13. https://doi.org/10.1145/1077464.1077466

[74] Lihi Zelnik-Manor and Pietro Perona. 2004. Self-Tuning Spectral Clustering. In *NIPS*. 1601–1608.

[75] Junwei Zhang, Min Gao, Junliang Yu, Lei Guo, Jundong Li, and Hongzhi Yin. 2021. Double-Scale Self-Supervised Hypergraph Learning for Group Recommendation. In *CIKM*. ACM, 2557–2567. https://doi.org/10.1145/3459637.3482426

[76] Shuqin Zhang and Hongyu Zhao. 2012. Community identification in networks with unbalanced structure. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 85, 6 (2012), 066114.

[77] Chen Zhe, Aixin Sun, and Xiaokui Xiao. 2019. Community Detection on Large Complex Attribute Network. In *KDD*. 2041–2049. https://doi.org/10.1145/3292500.3330721

[78] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. 2006. Learning with Hypergraphs: Clustering, Classification, and Embedding. In *NeurIPS*. 1601–1608. https://doi.org/10.7551/mitpress/7503.003.0205

[79] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2009. Graph Clustering Based on Structural/Attribute Similarities. *Proc. VLDB Endow.* 2, 1 (2009), 718–729. https://doi.org/10.14778/1687627.1687709

[80] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2010. Clustering Large Attributed Graphs: An Efficient Incremental Approach. In *2010 IEEE International Conference on Data Mining*. 689–698. https://doi.org/10.1109/ICDM.2010.41

[81] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2020. Deep Graph Contrastive Representation Learning. *CoRR* abs/2006.04131 (2020). arXiv:2006.04131 https://arxiv.org/abs/2006.04131

Dataset	TCL+ p_d	TCL+ w_s	TCL+ learning rate	GRC+ p_e	GRC+ p_a	GRC+ learning rate
C13	0.9	2^1	5e-04	0.8	0.1	5e-03
WIK	0.7	2^{-4}	5e-04	0.9	0.3	5e-03
COA	0.9	2^1	1e-04	0.6	0.0	1e-03
COC	0.7	2^{-1}	5e-04	0.6	0.1	5e-03
CIC	0.9	2^1	5e-05	0.0	0.2	1e-03
NEW	0.7	2^4	5e-03	0.1	0.2	5e-03
PBC	0.9	2^1	5e-03	0.8	0.6	5e-03
DBA	0.9	2^4	5e-03	\	\	\

Table 9: Hyperparameter settings on datasets

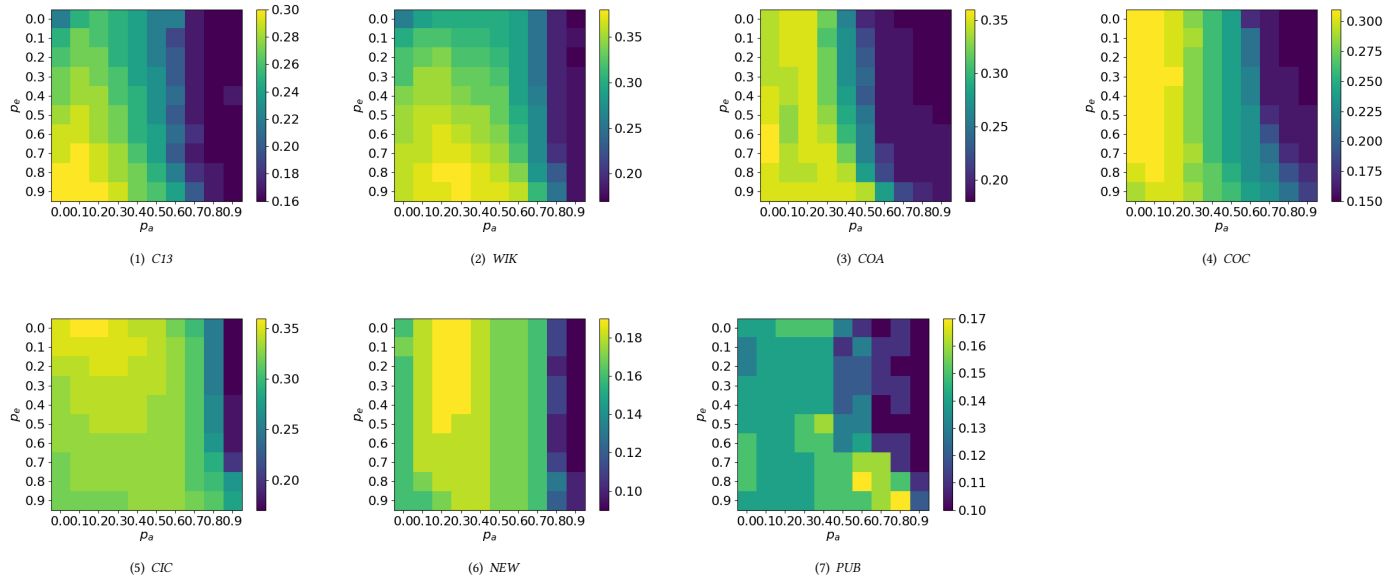
A Hyperparameters

In this section, we perform sensitivity analysis on hyperparameters in TCL+ and GRC+ for tuning. Following [43, 61, 81], we conduct attributed hypergraph clustering on all tested datasets, varying hyperparameter values through a grid search. Clustering quality is evaluated based on validation performance using F-measure. We choose the hyperparameters that yield the best performance and use these hyperparameters on the test set. The chosen hyperparameter values are summarized in Table 9.

For GRC+, we investigate the impact of p_e and p_a, which determine the portions of edges and node attributes to be removed, respectively. Figure 10 reports the F-measure scores with p_e and p_a values ranging from 0.0 to 0.9 in increments of 0.1. Figure 11 shows the F-measure scores with the learning rate set to values from [1e−5, 5e−5, 1e−4, 5e−4, 1e−3, 5e−3].

For TCL+, we investigate the impact of p_d, which determines the probability that an entry in $\mathbf{S}$ is set to zero, and w_s. Figure 12 reports the F-measure scores on datasets with p_d values ranging from 0.1 to 0.9 in increments of 0.1, and w_s values chosen from [2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^0, 2^1, 2^2, 2^3, 2^4]. Figure 13 shows the F-measure scores with the learning rate set to values from [1e−5, 5e−5, 1e−4, 5e−4, 1e−3, 5e−3].

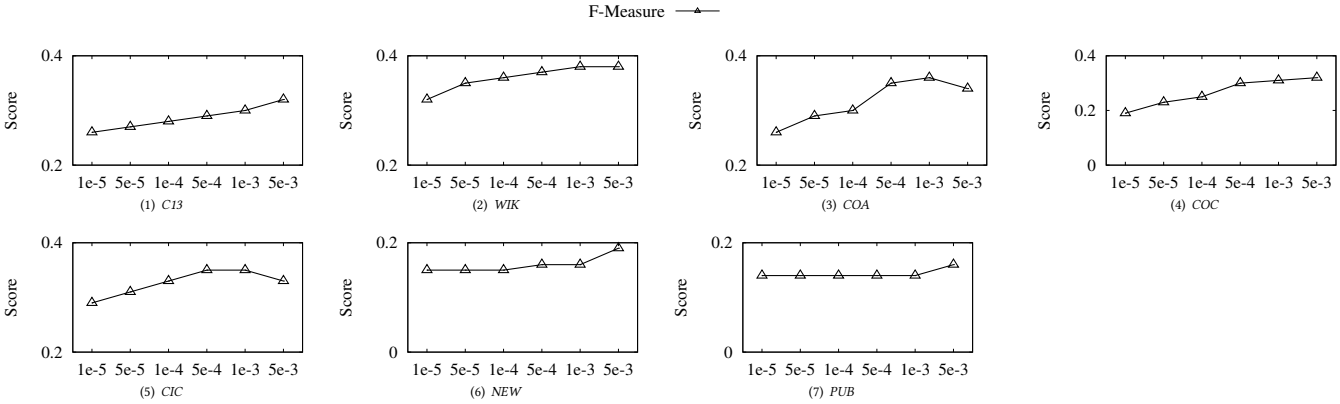
Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009



(1) *C13* (2) *WIK* (3) *COA* (4) *COC*

(5) *CIC* (6) *NEW* (7) *PUB*

Figure 10: Sensitivity: Clustering Quality of GRC+ on Varying Hyperparameters p_e and p_a in terms of F-measure



(1) *C13* (2) *WIK* (3) *COA* (4) *COC*

(5) *CIC* (6) *NEW* (7) *PUB*

Figure 11: Sensitivity: Clustering Quality of GRC+ on Varying Learning Rate

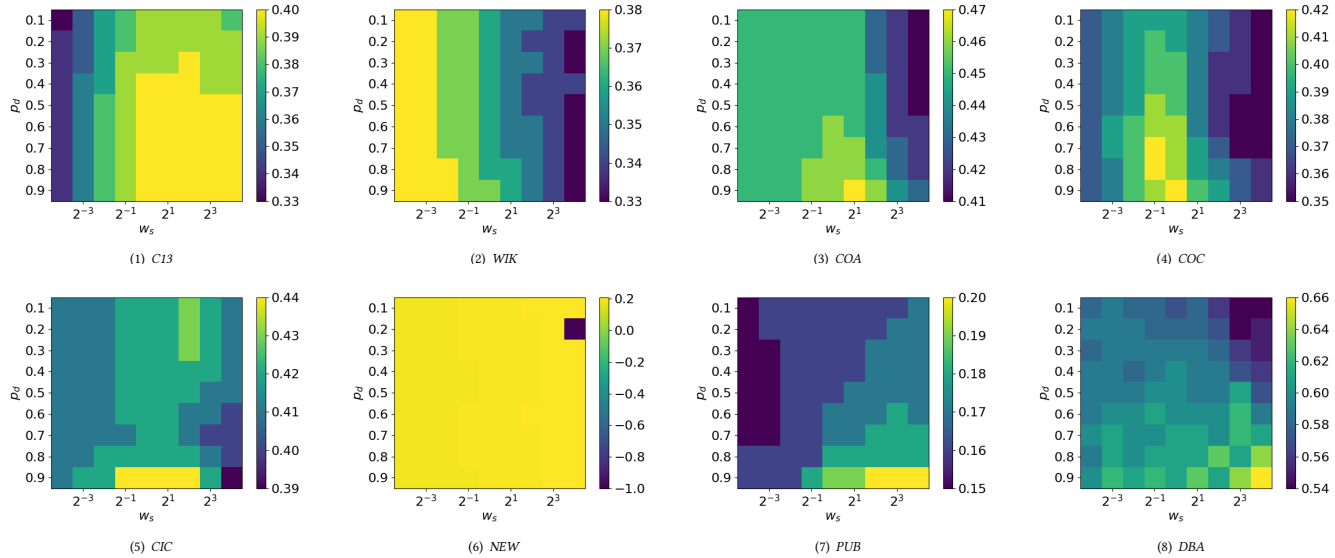


Figure 12: Sensitivity: Clustering Quality of TCL+ on Varying Hyperparameters p_d and w_s in terms of F-measure and ARI

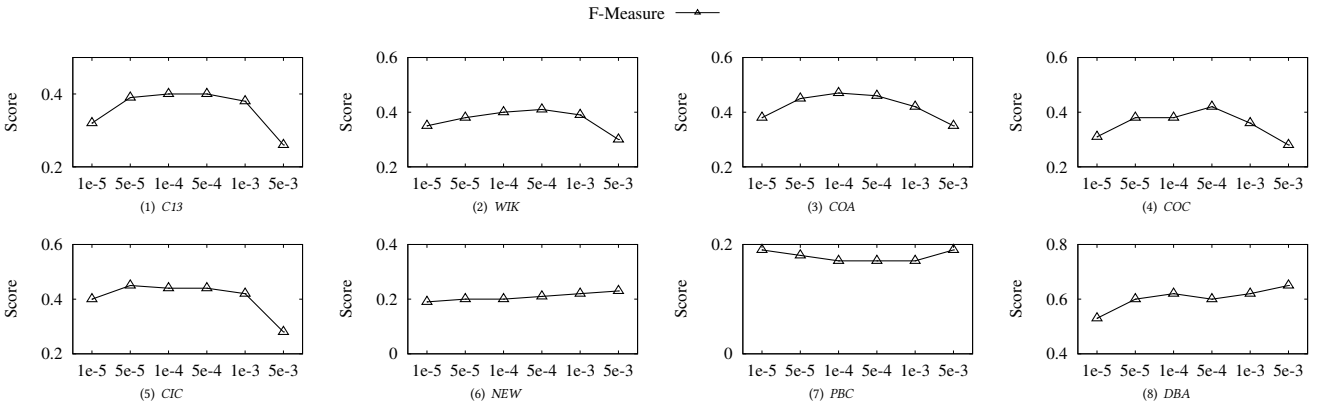


Figure 13: Sensitivity: Clustering Quality of TCL+ on Varying Learning Rate