ADVANCING CLUSTERING AND EMBEDDING FOR ATTRIBUTED NETWORK STRUCTURES

YIRAN LI

PhD

The Hong Kong Polytechnic University

2025

The Hong Kong Polytechnic University

Department of Computing

Advancing Clustering and Embedding for Attributed Network Structures

Yiran Li

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

April 2025

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgment has been made in the text.

Signature: ____________________

Name of Student: Yiran Li

Abstract

Attributed network structures, encompassing graphs, hypergraphs, and multi-view graphs, are fundamental in modeling complex systems across domains like social networks, bioinformatics, and e-commerce. However, existing clustering and embedding methods often struggle to capture complex network structures and scale for big data, limiting their effectiveness. This thesis advances the analysis of attributed network structures by proposing novel approaches that integrate structural and attribute information to achieve high-quality, efficient, and scalable solutions for clustering and embedding.

The first contribution introduces `ANCKA`, a versatile clustering framework that leverages K-nearest neighbor augmentation to partition nodes across attributed graphs, hypergraphs, and multiplex graphs. By efficiently optimizing a novel objective based on random walk, `ANCKA` delivers superior clustering performance. Building on this, the second contribution presents `SAHE`, an efficient embedding method for attributed hypergraphs, which unifies the computation of node and hyperedge embeddings to preserve multi-hop relationships. `SAHE` enhances quality and scalability through innovative similarity measures and approximation techniques. Finally, the third contribution develops `SGLA` and `SGLA+`, spectrum-guided algorithms for clustering and embedding multi-view attributed graphs. These algorithms cohesively integrate multiple graph and attribute views, achieving exceptional performance and efficiency.

Through extensive experiments on diverse real-world datasets, these frameworks demonstrate significant improvements over numerous baselines, often outperforming competitors by orders of magnitude in efficiency while producing high-quality results. Collectively, this thesis bridges critical gaps in effectiveness, efficiency, and scalability, enabling potential applications in community detection, bioinformatics modeling, and recommendation systems. By providing open-source implementations, including GPU-accelerated variants, this work lays a foundation for future advancements in attributed network analysis, fostering impactful solutions for complex network systems.

Publications Arising from the Thesis

1. Yiran Li, Renchi Yang, Jieming Shi, "Efficient and Effective Attributed Hypergraph Clustering via K-Nearest Neighbor Augmentation", in *Proceedings of the ACM on Management of Data*, 1.2, 116:1–116:23 (2023).

2. Yiran Li, Gongyao Guo, Jieming Shi, Renchi Yang, Shiqi Shen, Qing Li, Jun Luo, "A versatile framework for attributed network clustering via K-nearest neighbor augmentation", in *The VLDB Journal* , 33(6), 1913-1943 (2024).

3. Yiran Li, Gongyao Guo, Jieming Shi, Sibo Wang, "Efficient Integration of Multi-View Attributed Graphs for Clustering and Embedding", in *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, 3863-3875 (2025).

4. Yiran Li, Gongyao Guo, Chen Feng, Jieming Shi, "Effective and Efficient Attributed Hypergraph Embedding on Nodes and Hyperedges", accepted to *Proceedings of the VLDB Endowment*, 2025.

Acknowledgments

First and foremost, I want to thank my supervisor, Dr. Jieming Shi, for being an insightful mentor and rigorous critic of my research. Your work ethic and enthusiasm have profoundly inspired my passion for research. I extend heartfelt thanks to my collaborators, Renchi Yang, Gongyao Guo, and Chen Feng, whose expertise and teamwork were invaluable. I also appreciate my co-supervisor, Prof. Qing Li, and the staff of the Department of Computing at The Hong Kong Polytechnic University for their support throughout my PhD program. To my dearest friends and group mates, your companionship and encouragement have been a constant source of strength. Finally, I am forever grateful to my parents for their love and care, which have sustained me every step of the way.

Table of Contents

Abstract **i**

Publications Arising from the Thesis **iii**

Acknowledgments **iv**

List of Figures **x**

List of Tables **xiii**

1 Introduction **1**

1.1 Background and Significance 1

1.2 Research Gaps and Motivation 2

1.3 Research Problem and Scope 4

1.4 Research Objectives and Approach 5

1.5 Thesis Structure 7

2 Related Work **8**

2.1 Attributed Network Clustering 8

2.2 Attributed Hypergraph Embedding . . . 11
2.3 Multi-view Attributed Graphs . . . 12

3 ANCKA: Attributed Network Clustering 14

3.1 Introduction . . . 15
3.2 Preliminaries . . . 19
3.3 Attributed Hypergraph Clustering . . . 21
3.3.1 KNN Augmentation . . . 23
3.3.2 (α, β, γ)-Random Walk . . . 25
3.3.3 Objective Function . . . 27
3.4 Theoretical Analysis for `AHCKA` . . . 28
3.5 The `AHCKA` Algorithm . . . 33
3.5.1 Main Algorithm . . . 33
3.5.2 Greedy Initialization of BCM . . . 36
3.5.3 Complexity . . . 38
3.6 The `ANCKA` framework . . . 39
3.6.1 Generalized (α, β, γ)-Random Walk . . . 39
3.6.2 `ANCKA` Algorithm . . . 41
3.7 GPU-Accelerated `ANCKA-GPU` . . . 44
3.8 Experiments . . . 48
3.8.1 Experimental Setup . . . 49
3.8.2 Performance Evaluation . . . 54

3.8.3 Experimental Analysis 64

3.8.4 Convergence Analysis 70

3.8.5 Runtime Analysis 73

3.8.6 Additional Experiments 75

3.9 Summary 82

4 SAHE: Attributed Hypergraph Embedding **84**

4.1 Introduction 84

4.2 Preliminaries 88

4.3 Similarities and Objectives 90

4.3.1 Attribute-Extended Hypergraph 91

4.3.2 Hypergraph Multi-Hop Node Similarity: HMS-N 93

4.3.3 Hypergraph Multi-Hop Hyperedge Similarity: HMS-E 97

4.3.4 A Base Method 100

4.4 The `SAHE` Method 103

4.4.1 Unify HMS-N and HMS-E Computations 103

4.4.2 HMS-N and HMS-E Approximations 106

4.4.3 `SAHE` Algorithm Details 109

4.5 Experiments 111

4.5.1 Experimental Setup 111

4.5.2 Node Classification 114

4.5.3 Hyperedge Link Prediction 115

4.5.4 Hyperedge Classification . . . 117
4.5.5 Embedding Efficiency . . . 117
4.5.6 Experimental Analysis . . . 119
4.6 Summary . . . 125

5 SGLA: Multi-view Attributed Graph Integration 127

5.1 Introduction . . . 127
5.2 Preliminaries and Problem Statement . . . 131
5.2.1 Preliminaries . . . 131
5.2.2 Problem Statement . . . 132
5.3 SGLA Objective . . . 134
5.3.1 Eigengap Objective . . . 134
5.3.2 Connectivity Objective . . . 137
5.3.3 The Full Objective . . . 138
5.4 Algorithms . . . 140
5.4.1 `SGLA` Method . . . 140
5.4.2 `SGLA+` Method . . . 143
5.5 Experiments . . . 148
5.5.1 Experimental setup . . . 148
5.5.2 Effectiveness and Efficiency on Clustering . . . 152
5.5.3 Effectiveness and Efficiency on Embedding . . . 154
5.5.4 Experimental Analysis . . . 156

5.6 Summary 161

6 Conclusion **162**

6.1 Summary of Contributions 162

6.2 Implications and Impact 164

6.3 Future Directions 165

6.4 Concluding Remarks 165

List of Figures

3.1 An Example Attributed Hypergraph 22

3.2 AAS and RCC on Cora-CA 23

3.3 Overview of `AHCKA` 32

3.4 Runtime of `AHCKA` with CPU parallelization 46

3.5 Varying K for AHC 65

3.6 Varying β for AHC 66

3.7 Varying K for AGC 66

3.8 Varying K for AMGC 67

3.9 Varying β for AGC 67

3.10 Varying β for AMGC 67

3.11 Varying γ on Attributed Hypergraphs. 68

3.12 Convergence Analysis, part 1 71

3.13 Convergence Analysis, part 2 72

3.14 Runtime breakdown of CPU-based `ANCKA` and `ANCKA-GPU` 73

4.1 An example of attributed hypergraph H. 90

4.2 Extended hypergraph $\mathcal{H}$. 91

4.3 Dual hypergraph $\mathcal{H}'$. 97

4.4 Overview of the `SAHE` algorithm. 102

4.5 Running time of generating node and hyperedge embeddings together 118

4.6 Scalability Test. 120

4.7 Varying K for `SAHE`. 121

4.8 Varying β for `SAHE`. 121

4.9 Varying r for `SAHE`. 121

4.10 Heatmaps between K, β, and r on Cora-CA for node classification. Darker shades indicate higher MiF1. Underlined results are reported in Table 4.3, while the optimal parameter combinations are in bold. . 122

4.11 Varying α. 125

4.12 Varying τ. 125

4.13 Varying b. 125

5.1 Multi-view attributed graph $\mathcal{G}$ with two graph views 128

5.2 A running example. 136

5.3 Plot of objective functions on Yelp. 140

5.4 Overview of `SGLA` and `SGLA+` algorithms. 142

5.5 Running time of clustering 151

5.6 Running time of embedding 155

5.7 Varying number of iterations t in `SGLA` for clustering accuracy 157

5.8 Varying ϵ for `SGLA`. 157

5.9 Varying γ for SGLA+. 157

5.10 Vary the number of weight vector samples in SGLA+. 158

5.11 Clustering accuracy with alternative integrations. 160

5.12 Embedding visualization on RM and Yelp 160

List of Tables

3.1 Dataset statistics (ANCKA) 49

3.2 Attributed Hypergraph Clustering (AHC) Quality on Small Datasets. 53

3.3 Attributed Hypergraph Clustering (AHC) Quality on Medium/Large Datasets. 54

3.4 Attributed Graph Clustering (AGC) Quality on Cora, Citeseer-UG & Wiki. 55

3.5 Attributed Graph Clustering (AGC) Quality on Citeseer-DG, Tweibo & Amazon2M. 55

3.6 Attributed Multiplex Graph Clustering (AMGC) Quality. 56

3.7 Efficiency of Attributed Hypergraph Clustering (AHC) 56

3.8 Efficiency of Attributed Graph Clustering (AGC) 58

3.9 Efficiency of Attributed Multiplex Graph Clustering (AMGC) 59

3.10 Evaluation between ANCKA and ANCKA-GPU. 61

3.11 Additional GPU baselines for AHC. 63

3.12 Additional GPU baselines for AGC. 63

3.13 Additional GPU baselines for AMGC. 63

3.14 Ablation Analysis on AHC 68

3.15 Ablation Analysis on AGC 69

3.16 Ablation Analysis on AMGC 69

3.17 Varying similarity measures in KNN construction for `ANCKA` on AHC. 74

3.18 Varying similarity measures in KNN construction for `ANCKA` on AGC. 74

3.19 Varying similarity measures in KNN construction for `ANCKA` on AMGC. 75

3.20 Objective values $f(\mathcal{C})$ achieved by a method and $f(\mathcal{C}^*)$ of ground truth for AHC 76

3.21 Objective values $f(\mathcal{C})$ achieved by a method and $f(\mathcal{C}^*)$ of ground truth for AGC 76

3.22 Objective values $f(\mathcal{C})$ achieved by a method and $f(\mathcal{C}^*)$ of ground truth for AMGC 77

3.23 Evaluation between CPU-based `ANCKA` and `ANCKA-GPU` 78

3.24 Impact of exerting absolute or relative similarity thresholds (ϵ_s or ϵ_p) on the KNN graph on AHC. 79

3.25 Impact of exerting absolute or relative similarity thresholds (ϵ_s or ϵ_p) on the KNN graph on AGC. 79

3.26 Impact of exerting absolute or relative similarity thresholds (ϵ_s or ϵ_p) on the KNN graph on AMGC. 79

3.27 Impact of conflicting attribute and network information on AHC performance of `ANCKA`. 80

3.28 Impact of conflicting attribute and network information on AGC performance of `ANCKA`. 80

3.29 Impact of conflicting attribute and network information on AMGC performance of `ANCKA`. 80

3.30 Clustering performance comparison between `FSSC` and `ANCKA` on AHC 81

3.31 Clustering performance comparison between `FSSC` and `ANCKA` on AGC 81

3.32 Clustering performance comparison between `FSSC` and `ANCKA` on AMGC 81

4.1 Frequently used notations (`SAHE`) 88

4.2 Dataset statistics. 112

4.3 Node classification performance 112

4.4 Hyperedge link prediction performance 114

4.5 Hyperedge classification performance 116

4.6 Approximation Error (MAE). 120

4.7 Ablation analysis of HMS-N on node classification performance. . . . 123

4.8 Ablation analysis of HMS-E on hyperedge classification performance. 123

4.9 Node classification performance for extended baselines. 123

4.10 Hyperedge link prediction performance for extended baselines. 123

4.11 Hyperedge classification performance for extended baselines. 123

5.1 Frequently used notations (`SGLA`) 133

5.2 Statistics of multi-view attributed graph datasets. 148

5.3 Clustering quality 149

5.4 Embedding performance for node classification 152

Chapter 1

Introduction

1.1 Background and Significance

Attributed networks have emerged as a powerful paradigm for modeling complex systems across diverse domains, from social interactions [115] to biological processes [129]. Unlike traditional networks that capture only the structure of relationships, an attributed network enriches its topology—whether pairwise edges or higher-order hyperedges—with attributes associated with nodes, offering a multi-dimensional representation of entities and their interactions. This thesis focuses on three representative types of attributed networks: attributed graphs, attributed hypergraphs, and multi-view attributed graphs, each presenting unique characteristics and analytical challenges. Attributed graphs feature pairwise edges connecting exactly two nodes, as seen in social networks or citation networks, where nodes carry attributes like user profiles or publication metadata. Attributed hypergraphs extend this model by introducing hyperedges, which connect an arbitrary number of nodes to capture multiway relationships—such as paper co-authorships or group purchases—while associating nodes with attributes like academic profiles or product descriptions. Multi-view attributed graphs (MVAGs) further generalize this concept, describing entities through multiple graph views (*e.g.*, personal versus professional connections) and attribute views (*e.g.*, numerical or categorical features), with attributed multiplex graphs representing a special case where a single attribute view accompanies multiple graph layers. These structures collectively enable a richer depiction of real-world systems, where relationships and attributes vary across perspectives and dimensions.

Clustering and embedding stand out as two pivotal tasks in network analysis, each holding significant value across a wide range of applications. Clustering aims to partition the nodes of an attributed network into disjoint groups based on structural connectedness and attribute similarity, a task explored in this thesis through novel frameworks for attributed hypergraphs [73] and extended to graphs and multiplex graphs [72]. Its significance lies in its ability to uncover hidden structures, such as communities in social networks, functional modules in biological networks, or cohesive groups in academic hypergraphs, supporting applications like community detection [139], image classification [11], and Web query analysis [123]. On the other hand, embedding maps nodes—and in some cases hyperedges—to points in a low-dimensional space that preserves their structural and attribute information, a challenge addressed here through an innovative approach for attributed hypergraphs and an integration scheme for multi-view attributed graphs. This task is crucial for enabling downstream tasks like recommendation systems [155], spam detection [68], and genomic expression modeling [5], where latent representations enhance predictive accuracy and interpretability. Together, these tasks—clustering and embedding—drive advancements in fields such as social network analysis [109], bioinformatics [149], and e-commerce [120] by providing systemic insights and informative representations.

1.2 Research Gaps and Motivation

Clustering and embedding are cornerstone tasks in attributed network analysis, critical for uncovering patterns and deriving representations in complex systems. However, despite their significance, existing methods exhibit substantial shortcomings that limit their performance in practical applications. A primary expectation is that clustering and embedding outcomes achieve high quality: clusters should reflect both the network's topology and attribute distributions, while embeddings should preserve structural and attribute characteristics in low-dimensional spaces. Yet, current approaches often fail to effectively integrate these dual aspects—structure and attributes—particularly for the intricate forms of attributed networks central to this thesis: graphs, hypergraphs, and multi-view attributed graphs. This thesis is motivated by the urgent need to address these gaps, developing innovative methods that deliver effective, efficient, and scalable solutions for clustering and embedding in such networks.

One major gap lies in the lack of tailored algorithms that capture the unique complexities of attributed networks. Traditional methods designed for simple graphs, such as METIS for graph clustering [55], or for numerical data, like k-means, are ill-suited for attributed networks, where structural relationships and attribute values represent fundamentally distinct data types. Applying these methods often yields suboptimal results, as they fail to model the interplay between topology and attributes. This challenge is amplified in more complex structures like attributed hypergraphs and multi-view attributed graphs. In hypergraphs, hyperedges encode higher-order relationships—such as co-authorships or co-purchases—that cannot be reduced to simple graphs without losing essential information [43]. Similarly, multi-view attributed graphs comprise multiple graph and attribute views, each with distinct semantics (e.g., social versus professional connections, or text versus visual features), which are compromised when treated as a single attributed graph [153, 119]. Despite the growing prevalence of these networks in domains like social analysis, bioinformatics, and Web query analysis, research on clustering and embedding methods specifically designed for their complexity remains limited.

Equally pressing is the issue of efficiency and scalability, as real-world attributed networks are often vast in scale. For instance, social networks like Meta's platforms host over 3.35 billion daily active users across Facebook, Instagram, and WhatsApp [85], while bioinformatics databases like STRING catalog over 20 billion protein-protein interactions among 59 million proteins. Effective analysis of such networks demands algorithms that operate with low computational overhead. However, many existing methods, particularly those leveraging graph neural networks (GNNs) for clustering and embedding [78, 59], require long training time and extensive GPU resources, rendering them impractical for large-scale applications [143]. This is particularly true for hypergraph embedding, where few efficient solutions exist to handle higher-order interactions [157], and for multi-view graph analysis, where integrating diverse views adds computational burdens [32].

These gaps—insufficient effectiveness in capturing network complexity and inadequate efficiency for large-scale settings—underscore the need for novel approaches. This thesis aims to bridge these deficiencies by proposing solutions that leverage both structural and attribute information across attributed graphs, hypergraphs, and multi-view graphs. By addressing the challenges of quality, efficiency, and scalability, our work seeks to advance the analysis of complex network systems, enabling robust applications in fields such as community detection, genomic expression modeling, and recommendation systems.

1.3 Research Problem and Scope

The analysis of attributed networks—spanning graphs, hypergraphs, and multi-view structures—represents a critical frontier in understanding complex systems, where entities are defined by both various relationships and rich attribute information. At the core of this domain lie two fundamental tasks: clustering and embedding. Clustering aims to group nodes into cohesive clusters that reflect both topological proximity and attribute similarity, revealing underlying patterns such as communities or functional modules. Embedding seeks to map nodes, and potentially higher-order structures like hyperedges, into low-dimensional spaces that preserve structural and attribute characteristics, enabling applications like recommendation systems and predictive modeling. However, achieving high-quality outcomes that incorporate the dual aspects of attributed networks remains a formidable challenge, particularly for those with higher-order interactions or multi-view data. This thesis addresses the long-term research mission of developing comprehensive, computationally viable frameworks for clustering and embedding in attributed networks, aiming to deliver solutions that are effective in capturing network complexity and scalable to meet the demands of real-world applications.

The scope of this problem is broad, encompassing the diverse forms of attributed networks and their analytical challenges. Attributed graphs require methods that integrate pairwise connections with attribute values, while hypergraphs demand approaches that preserve multiway relationships encoded by hyperedges. Multi-view attributed graphs introduce additional complexity, necessitating the synthesis of multiple graph and attribute views with distinct semantics. This thesis positions clustering and embedding as a unified research agenda, with the long-term vision of establishing generalizable methodologies that advance network analysis across domains like social network analysis, bioinformatics, and e-commerce, addressing both current limitations and future scalability needs.

1.4 Research Objectives and Approach

To advance the long-term goal of developing clustering and embedding frameworks, this thesis pursues three specific objectives, each addressing a distinct challenge within the scope of attributed network analysis. These objectives correspond to three interconnected contributions, each targeting a specific problem while collectively contributing to the broader research agenda:

1. **Clustering for Diverse Attributed Networks**: We aim to develop a clustering framework that effectively partitions nodes across attributed graphs, hypergraphs, and multiplex graphs, addressing the challenge of integrating structural and attribute information in diverse network types [72]. Building upon a preliminary work on attributed hypergraph clustering [73], this framework tackles the problem of capturing higher-order interactions and homogeneous attributes, ensuring clusters align with both topology and attribute distributions.

2. **Attributed Hypergraph Node and Hyperedge Embedding**: We seek to create an embedding method for attributed hypergraphs that generates low-dimensional representations of nodes and hyperedges, preserving multiway relationships and attribute characteristics. This objective addresses the scarcity of efficient embedding techniques for higher-order networks, a critical gap in representation learning for applications like genomic expression modeling.

3. **Unified Clustering and Embedding for Multi-View Attributed Graphs**: We propose an integration scheme that unifies clustering and embedding tasks for multi-view attributed graphs, synthesizing multiple graph and attribute views to produce cohesive outcomes. This work confronts the complexity of multi-view data, enabling high-quality analysis for applications such as recommendation systems and image processing.

Our research approach is centered on designing innovative algorithms that prioritize both effectiveness and efficiency. We leverage techniques such as attribute-based graph augmentation to enable the representation of node attributes in networks, whether homogeneous or multi-view, and structural similarity measures to capture topological relationships, from pairwise edges to hyperedges. In addition to developing solutions that address specific network types, we also work on adaptability to varying complexities, such as extending from higher-order interactions in hypergraphs to simple pairwise edges in graphs or semantically diverse multi-view graphs. Efficiency is a key focus, with methods designed to minimize computational overhead, making them viable for large-scale networks like those in social media or bioinformatics. This thesis integrates these contributions into a cohesive narrative, demonstrating how each work advances the goal of attributed network clustering and embedding, paving the way for scalable, impactful network analysis.

1.5 Thesis Structure

The thesis is organized as follows to present a coherent narrative of our research. Chapter 2 provides a comprehensive review of related work, establishing the theoretical and methodological context for our contributions. Chapter 3 introduces our extended clustering framework, including the attributed hypergraph clustering method, detailing its design and performance. Chapter 4 explores our attributed hypergraph embedding approach, focusing on its ability to capture higher-order relationships. Chapter 5 presents the integration scheme for clustering and embedding in multi-view attributed graphs, highlighting its unified methodology. Finally, Chapter 6 synthesizes the findings, discusses their implications, and outlines future research directions.

Chapter 2

Related Work

This chapter reviews prior research relevant to the clustering and embedding of attributed network structures, providing the foundation for the contributions presented in this thesis. Section 2.1 surveys clustering methods for simple hypergraphs, attributed hypergraphs, attributed graphs, and multiplex graphs, highlighting the need for a unified framework to address their diverse requirements. Section 2.2 examines embedding techniques applicable to attributed hypergraphs, identifying limitations in their quality and scalability. Section 2.3 focuses on multi-view attributed graphs (MVAGs), reviewing the state of clustering and embedding methods for these structures. Together, these sections establish the context for our novel approaches developed in Chapter 3 (`ANCKA`), Chapter 4 (`SAHE`), and Chapter 5 (`SGLA` and `SGLA`+), respectively.

2.1 Attributed Network Clustering

Hypergraph Clustering. Motivated by the applications in circuit manufacturing, partitioning algorithms have been developed to divide hypergraphs into partitions/clusters, such as `hMetis` [54] and `KaHyPar` [105]. These methods typically adopt a three-stage framework consisting of coarsening, initial clustering, and refinement stages. These algorithms directly perform clustering on a coarsened hypergraph with a relatively small size. In addition, they run a portfolio of clustering algorithms and select the best outcome. These algorithms rely on a set of clustering heuristics and lack the extensibility for exploiting node attribute information. Hypergraph Normalized Cut (HNCut) [156] is a conductance measure for hypergraph clusters from which the normalized hypergraph Laplacian $\Delta = \mathbf{I} - \Theta$ is derived for spectral clustering, where $\Theta = \mathbf{D}_V^{-1/2}\mathbf{H}^T\mathbf{D}_E^{-1}\mathbf{H}\mathbf{D}_V^{-1/2}$. Alternatively, `hGraclus` [123] optimizes the HNCut objective using a multi-level kernel K-means algorithm. Non-negative matrix factorization has also been applied to hypergraph clustering [41]. Despite the theoretical soundness, these algorithms are less efficient than the aforementioned partitioning algorithms and they do not utilize node attributes either. For the problem of hypergraph local clustering, which is to find a high-quality cluster containing a specified node, a sweep cut method is proposed [113] to find the cluster based on hypergraph Personalized PageRank (PPR) values. In this thesis, we focus on global clustering, a different problem from local clustering.

Attributed Hypergraph Clustering. There exist studies designing dedicated clustering algorithms on attributed hypergraphs. `JNMF` [23] is an AHC algorithm based on non-negative matrix factorization (NMF). With normalized hypergraph Laplacian [156] matrix $\Delta = \mathbf{I} - \Theta$ and attribute matrix $\mathbf{X}$, `JNMF` optimizes the following joint objective that includes a basic NMF part as well as a symmetric NMF part: $\min_{\mathbf{W},\mathbf{M},\tilde{\mathbf{M}}\geq 0} ||\mathbf{X} - \mathbf{W}\mathbf{M}||_F^2 + \alpha||\Theta - \tilde{\mathbf{M}}^\mathsf{T}\mathbf{M}||_F^2 + \beta||\tilde{\mathbf{M}} - \mathbf{M}||_F^2$. With optimization using block coordinate descent (BCD) scheme, the matrix $\mathbf{M}$ is expected to encode cluster memberships. `MEGA` [123] extends the formulation of `JNMF` clustering objective for *semi-supervised* clustering of multi-view data containing hypergraph, node attributes as well as pair-wise similarity graph. `MEGA`'s clustering performance is further enhanced by initialization with `hGraclus` algorithm. `GNMF` [10] algorithm is originally proposed for high dimensional data clustering, while the authors of [27] extend its objective with the hypergraph normalized Laplacian [156] so that it spawns baseline methods for AHC. Although NMF-based algorithms sometimes produce clusters of good quality, their scalability is underwhelming, as shown in our experiments in Section 3.8. As the state-of-the-art algorithm for attributed hypergraph clustering, `GRAC` [27] performs hypergraph convolution [136] on node attributes, which resembles the hypergraph diffusion process with mediators [12]. Then clusters are predicted from the propagated features via a spectral algorithm.

Attributed Graph Clustering. There exists a collection of studies on attributed graph clustering. Some studies perform attributed graph clustering by adopting probabilistic models to combine graph structure with attributes, including discriminative models such as `PCL-DC` [145] and generative models such as `BAGC` [134]. Nevertheless, these methods are typically limited to handling categorical attributes. Moreover, inference over the probability distribution of $O(2^n)$ hyperedges poses a significant challenge against their generalization to hypergraph. `GNMF` [10] is an NMF-based algorithm that enhances performance by modifying the Laplacian regularizer used in traditional NMF to utilize the Laplacian matrix constructed from the graph structure. Within the random walk framework, `SA-Cluster` [159] algorithm augments the original graph with virtual nodes representing each possible attribute-value pair and performs k-Medroids clustering using a random walk distance measure. `ACMin` [144] defines attributed random walk by adding virtual attribute nodes as bridges and combines it with graph random walk into a joint transition matrix. In a fashion similar to GCN [61], `AGCGCN` [154] performs graph convolution on node attributes to produce smooth feature representations that incorporate network structure information and subsequently applies spectral clustering. For their spectral algorithm, the authors also design heuristics to prevent propagated features from over-smoothing that undermines cluster quality. `GRACE` [52] adopts graph convolution on node attributes to fuse all available information and perform a spectral algorithm based on `GRAC` [27]. `FGC` [53] exploits both node features and structure information via graph convolution and applies spectral clustering on a fine-grained graph that encodes higher-order relations.

Attributed Multiplex Graph Clustering. Via unsupervised learning on attributed multiplex graphs, neural network models can learn node embeddings for clustering, e.g., `O2MAC` [24] and `HDMI` [49]. `GRACE` [52] constructs a multiplex graph Laplacian and uses this matrix for graph convolution. Other methods find a single graph that encodes the node proximity relations in all graph layers and attributes. `MCGC` [91] performs graph filtering on attributes and learns a consensus graph leveraging contrastive regularization, while `MAGC` [77] exploits higher-order proximity to learn consensus graphs without deep neural networks.

2.2 Attributed Hypergraph Embedding

Existing embedding methods struggle to support attributed hypergraphs natively while scaling efficiently for massive data, particularly in the context of attributed hypergraph node and hyperedge embedding (AHNEE), where embeddings are demanded for both nodes and hyperedges. Early hypergraph embedding efforts, like [156], use the Laplacian matrix spectrum for node embeddings, focusing on clustering but neglecting attributes and long-range connectivity essential for AHNEE 's relational closeness. Methods extending node2vec [36] to hypergraphs, such as `Hyper2vec` [45] and its dual-enhanced version [44], capture long-ranged relations via random walks, yet omit attribute information and hyperedge embeddings, limiting their applicability for AHNEE. Another approach, [147], models hyperedges as multi-linear products of node embeddings, but lacks attribute consideration and scalability.

The emergence of hypergraph neural networks also enabled a multitude of recent approaches. `TriCL` [64] uses contrastive learning with augmentations to embed nodes, `HypeBoy` [60] masks attributes and designs a hyperedge-filling task for self-supervised node embeddings, and [22] applies GNNs on an expanded graph with cluster-based loss. Nevertheless, these methods focus solely on nodes, incur high training costs (e.g., $O(n^2)$ or worse), and lack hyperedge embedding support, rendering them unsuitable for scalable AHNEE. `VilLain` [65] formulates the self-supervision as a label propagation process while ignoring node attributes and incurring high training costs. There are a few specialized methods that learn node representations for certain hypergraphs, like [138] for location-based networks, [130] for user-item recommendations, [133] for trust relations, and [74] for spatio-temporal crime data, but their targeted design for domain-specific data and limited scalability hinder the applicability for general attributed hypergraphs.

Alternatively, embedding techniques designed for graphs or bipartite graphs can be adapted for attributed hypergraphs by reducing the hyperedge structures into pairwise edges. With each hyperedge converted to a fully connected subgraph via clique-expansion, hypergraphs can be processed by graph embedding methods (`NetMF` [99], `STRAP` [146], `LightNE` [98]) or attributed graph embedding method [143] based on matrix factorization. On the other hand, star-expansion of hypergraphs results in dense edges between two sets of nodes, enabling bipartite graph embedding methods (`BiANE` [47], `AnchorGNN` [128]). However, these transformations weaken higher-order connections critical to AHNEE while producing dense graphs with high complexity, resulting in compromised embedding quality and efficiency, as our experiments show.

2.3 Multi-view Attributed Graphs

For basic attributed graphs with one graph and one attribute view, attributed network embedding and clustering have been extensively studied in the literature [158, 134, 87, 143, 79, 142]. For instance, Bayesian probabilistic model [134] and graph auto-encoder [87] have been adopted for clustering. An attributed graph embedding algorithm [143] captures the multi-hop affinity between nodes and attributes [142].

GNN-based embedding method AnECI [79] strengthens the robustness by preserving communities, while CONN [114] adopts selective graph diffusion with attribute augmentation. In [75], the authors propose using diverse pretext tasks to capture different signals in graphs with heterophily into embeddings. These approaches do not consider the unique characteristics of MVAGs and tend to yield suboptimal performance.

On MVAGs, MCGC [91] and MAGC [77] construct a consensus graph matrix for MVAG clustering in $O(n^2)$ time where n is the number of nodes, by optimizing a dense $n \times n$ adjacency matrix that minimizes reconstruction loss on each view. MvAGC [76] improves their complexity to linear time with node sampling while compromising result quality and stability. Their problem formulations neglect the overall structure of $\mathcal{G}$ and suffer from the difficulty of optimizing at least $O(n)$ variables. Besides, various GNNs have been adopted for MVAGs, including O2MAC [24], MAGCN [14] and DMG [86]. For instance, the clustering model MAGCN uses graph auto-encoders to map each view to latent representations for reconstruction. [58] combines graph views by fusing Laplacian matrices and trains a semi-supervised GNN. Other studies [92, 49, 152] adopt mutual information models to learn view-specific node embeddings and aggregate them with attention mechanism. These methods incur high costs of training and exhibit inferior performance on MVAGs. MEGA [123] tackles semi-supervised MVAG clustering by joint nonnegative matrix factorization. 2CMV [82] learns the consensus and complementary components from each view via matrix factorization with $O(n^2)$ complexity. LMGEC [30] addresses clustering and embedding within a unified formulation, while the embedding quality is inferior to its clustering performance.

There are also algorithms that only handle attribute views for clustering, as surveyed in [25]. For instance, a work [160] adopts a weighting objective to minimize the subspace distances between its integration result and each view. A recent study [135] learns a robust fused representation of noisy attributes via meta-learning. However, these methods [160, 57, 135] do not consider graph topological properties of MVAGs.

Chapter 3

ANCKA: Attributed Network Clustering

This chapter presents ANCKA [72], a versatile framework for clustering attributed networks, marking the first main technical contribution to the thesis's goal of advancing clustering and embedding for attributed network structures. Addressing the challenge of partitioning nodes across diverse types of attributed network—graphs, hypergraphs, and multiplex graphs—ANCKA integrates structural and attribute information to deliver effective and scalable solutions, as motivated by the gaps in existing methods outlined in Chapter 2. Through its novel clustering framework for attributed hypergraphs, extended to graphs and multiplex graphs, ANCKA introduces KNN augmentation as a key contribution, providing a critical methodological basis for the embedding and multi-view integration approaches in Chapters 4 and 5, thereby advancing the analysis of complex network structures.

3.1 Introduction

An attributed network contains a network topology with attributes associated with nodes. Representative types of attributed networks include attributed graphs, attributed hypergraphs, and attributed multiplex graphs. Given an attributed network $\mathcal{N}$, node clustering is an important task in graph mining, which aims to divide the n nodes of $\mathcal{N}$ into k disjoint clusters, such that nodes within the same cluster are close to each other in the network topology and similar to each other in terms of attribute values. Clustering on attributed networks finds important applications in biological analysis [23], online marketing [134], social network analysis [139, 107], Web analysis [123], image processing [11], etc.

In this work, we present `ANCKA` [72], an effective and efficient attributed network clustering method that is versatile to support attributed hypergraph clustering (AHC), attributed graph clustering (AGC), and attributed multiplex graph clustering (AMGC). `ANCKA` builds upon the `AHCKA` [73] algorithm, which we originally developed for AHC and is also covered in this chapter. In what follows, we first elaborate on AHC and then generalize to AGC and AMGC.

In a hypergraph, each edge can join an arbitrary number of nodes, referred to as a *hyperedge*. The hyperedge allows a precise description of multilateral relationships between nodes, such as collaboration relationships of multiple authors of a paper, interactions among proteins [34], products purchased together in one shopping cart, transactions involving multiple accounts [126]. In practice, nodes in hypergraphs are often associated with many attributes, e.g., the academic profile of authors and the descriptive data of products. The AHC problem is to divide the n nodes in such an attributed hypergraph into k disjoint clusters such that nodes within the same cluster are close to each other with high connectedness and homogeneous attribute characteristics. AHC finds numerous real-life applications in community discovery [46], organization structure detection [23], Web query analysis [123], image classification [11], biological analysis [127], etc. As another example, AHC can cluster together academic publications with high relevance by considering co-authorship hyperedges and keyword attributes in academic hypergraphs [27].

Effective AHC computation is a highly challenging task, especially for large attributed hypergraphs with millions of nodes. First, nodes, hyperedge connections, and attributes are heterogeneous objects with inherently different traits, whose information cannot be seamlessly integrated in a simple and straightforward way. Second, as observed in previous works on simple graphs [159, 144], higher-order relationships between nodes and node-attribute associations are crucial for clustering. However, computing such multi-hop relationships and associations via hyperedges usually with more than two nodes in attributed hypergraphs is rather difficult due to the complex hypergraph structures and prohibitive computational overheads (up to $O(n^2)$ in the worst case).

In the literature, a plethora of clustering solutions [105, 41, 62] are developed for plain hypergraphs. These methods overlook attribute information, leading to severely compromised AHC result quality. Besides, a large body of research on attributed graph clustering is conducted, resulting in a cornucopia of efficacious techniques [134, 144]. However, most of these works cannot be directly applied to handle large attributed hypergraphs with more complex and unique structures. Inspired by the technical advances in the above fields, a number of efforts have been made towards AHC computation in the past years. The majority of AHC methods rely on non-negative matrix factorization [23, 123], which requires numerous iterations of expensive matrix operations and even colossal space costs of materializing $n \times n$ dense matrices. Particularly, none of them take into account the higher-order relationships between nodes, thereby limiting their result utility. The state-of-the-art approach `GRAC` [27] extends *graph convolution* [61] to hypergraphs, indirectly incorporating higher-order relationships of nodes and attributes for clustering. Notwithstanding its enhanced clustering quality, `GRAC` runs in $O(n^2)$ time as an aftermath from costly graph convolution and SVD operations, which is prohibitive for large hypergraphs. To recapitulate, existing AHC approaches either yield sub-optimal clustering results or incur tremendous computational costs, rendering them impractical to cope with large attributed hypergraphs with millions of nodes.

Given the above, can we combine and orchestrate hypergraph topology and attribute information in an optimized way for improved clustering quality while achieving high scalability over large attributed hypergraphs? We offer a positive answer by presenting AHCKA (Attributed Hypergraph Clustering via K-nearest neighbor Augmentation), a novel AHC approach that significantly advances the state of the art in AHC computation. AHCKA surpasses existing solutions through several key techniques. The first one is a K-nearest neighbor (KNN) augmentation scheme, which augments the original hypergraph structure with a KNN graph containing additional connections constructed by adjacent nodes with K highest attribute similarities. This is inspired by a case study on a real dataset manifesting that incorporating all-pairwise node connections via attributes or none of them jeopardizes the empirical clustering quality. Second, AHCKA formulates the AHC task as a novel optimization problem based on a joint random walk model that allows for the seamless combination of high-order relationships from both the hypergraph and KNN graph. Further, AHCKA converts the original NP-hard problem into an approximate matrix trace optimization and harnesses efficient matrix operations to iteratively and greedily search for high-quality solutions. Lastly, AHCKA includes an effective initialization method that considerably facilitates the convergence of the optimization process using merely a handful of iterations. We conduct extensive experiments on attributed hypergraph data in different domains. Compared with baselines, AHCKA exhibits superior performance in both clustering quality and efficiency. For instance, on the Amazon dataset with 2.27 million nodes, AHCKA gains over 10-fold speedup and a significant improvement of 4.8% in clustering accuracy compared to state-of-the-art. Our work AHCKA has been published in [73].

In addition to attributed hypergraphs, attributed graphs and attributed multiplex graphs are prevalent in real-world scenarios, such as social networks [95] and citation networks [96]. Different from hypergraphs that allow more than two nodes to form an edge, in a graph, an edge connects exactly two nodes. A multiplex graph consists of multiple layers of graphs with a shared set of nodes, and different graph layers represent node connections from different perspectives or domains, e.g., different types of relationships or relations formed in different time frames or spaces [95, 96]. Attributed graph clustering (AGC) is one of the most significant graph mining problems, extensively studied in the literature [134, 144], with many applications, e.g., community detection in social networks [31] and functional cartography of metabolic networks [37]. Furthermore, a rich collection of studies on attributed multiplex graph clustering (AMGC) also exists in [91, 77, 24, 49], to support important applications, e.g., biological analysis [96], community detection [95] and social analysis [20]. A previous general framework [52] relies on expensive graph convolutions to support various clustering tasks.

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

We summarize the contributions of this work as follows:

- We devise a KNN augmentation scheme that exploits attributes to augment the original hypergraph structure in a cost-effective manner.
- We formulate the AHC task as an optimization with the objective of optimizing a quality measure based on a joint random walk model over the KNN augmented hypergraph.
- We propose a number of techniques for efficient optimization of the objective, including a theoretically-grounded problem transformation, a greedy iterative framework, and an effective initialization approach that drastically reduces the number of iterations till convergence.
- We justify the application of KNN augmentation to various types of networks, generalize the techniques, and design a versatile method `ANCKA` to efficiently perform AHC, AGC, and AMGC and produce high-quality clusters.
- We develop `ANCKA-GPU` with customized GPU kernels to improve the efficiency further with a series of GPU-based optimizations while maintaining clustering quality.
- The excellent performance of `ANCKA` is validated by comprehensive experiments against 19 AHC competitors, 16 AGC competitors, and 16 AMGC competitors, over real-world datasets.

3.2 Preliminaries

Attributed Network. Let $\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ be an attributed network, where $\mathcal{V}$ is the node set with cardinality $|\mathcal{V}| = n$, $\mathcal{E}$ is the edge (or hyperedge) set with cardinality $|\mathcal{E}| = m$, and $\mathbf{X} \in \mathbb{R}^{n \times d}$ represents a node attribute matrix. A node $v_j \in \mathcal{V}$ has degree $\delta(v_j)$, which is the number of edges (or hyperedges) incident to v_j. Each node v_j in $\mathcal{V}$ is associated with a d-dimensional attribute vector, denoted as $\mathbf{X}[j]$, i.e., the j-th row of the node attribute matrix $\mathbf{X}$. We consider three types of attributed networks $\mathcal{N}$, including attributed hypergraphs $\mathcal{H}$, attributed graphs $\mathcal{G}$, and attributed multiplex graphs $\mathcal{G}_M$, characterized by different nature of $\mathcal{E}$.

Attributed Hypergraph is denoted by $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$. $\mathcal{E}$ is the set of m hyperedges where each $e_i \in \mathcal{E}$ is a subset of $\mathcal{V}$ containing at least two nodes. A hyperedge e_i is said to be incident with a node v_j if $v_j \in e_i$. We denote by $\mathbf{H} \in \mathbb{R}^{m \times n}$ the incidence matrix of hypergraph $\mathcal{H}$, where each entry $\mathbf{H}[i, j] = 1$ if $v_j \in e_i$, otherwise $\mathbf{H}[i, j] = 0$. Let diagonal matrices $\mathbf{D}_V \in \mathbb{R}^{n \times n}$ and $\mathbf{D}_E \in \mathbb{R}^{m \times m}$ represent the degree matrix and hyperedge-size matrix of $\mathcal{H}$, where the diagonal entry $\mathbf{D}_V[j, j] = \delta(v_j)$ for $v_j \in \mathcal{V}$ and $\mathbf{D}_E[i, i] = |e_i|$ for $e_i \in \mathcal{E}$, respectively. Figure 3.1 shows an attributed hypergraph $\mathcal{H}$ with 8 nodes and 5 hyperedges, where each node has an attribute vector and hyperedges e_1, e_2 contain 4 and 3 nodes, i.e., $\{v_1, v_2, v_4, v_5\}$ and $\{v_1, v_3, v_4\}$, respectively.

Attributed Graph is denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where every edge in $\mathcal{E}$ connects exactly two nodes. A graph $\mathcal{G}$ can be undirected or directed. An undirected edge can be viewed as two directed edges of the same node pair in reversed directions. Different from a hypergraph incident matrix between nodes and hyperedges, graph adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ encodes the structure of $\mathcal{G}$, where entry $\mathbf{A}[i, j]$ is 1 if there is an edge from node v_i to node v_j, i.e., $(v_i, v_j) \in \mathcal{E}_G$, or 0 if otherwise. Let $\mathbf{D} \in \mathbb{R}^{n \times n}$ be the diagonal node degree matrix of $\mathcal{G}$.

Attributed Multiplex Graph is $\mathcal{G}_M = (\mathcal{V}, \mathcal{E}_1, ..., \mathcal{E}_L, \mathbf{X})$, consisting of L graph layers. Every l-th layer has its own edge set $\mathcal{E}_l$, and can be viewed as an attributed graph $\mathcal{G}_l$ with $\mathcal{E}_l$, adjacency matrix $\mathbf{A}_l$, and diagonal node degree matrix $\mathbf{D}_l$.

The Clustering Problem. Given an attributed network $\mathcal{N}$ that can be $\mathcal{H}$, $\mathcal{G}$, or $\mathcal{G}_M$, we study the clustering problem that encompasses *attributed hypergraph clustering* (AHC), *attributed graph clustering* (AGC), and *attributed multiplex graph clustering* (AMGC). Given a specified number k of clusters and an attributed network $\mathcal{N}$, the clustering task is to divide the node set $\mathcal{V}$ into k disjoint subsets $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ such that $\bigcup_{i=1}^{k} \mathcal{C}_i = \mathcal{V}$ and the following properties are satisfied:

1. Nodes within the same cluster are closely connected to each other in the network structure, while nodes in different clusters are far apart (**structure closeness**);

2. Nodes in the same cluster have similar attribute values, while nodes in different clusters vary significantly in attribute values (**attribute homogeneity**).

For instance, when the input network $\mathcal{N}$ is the attributed hypergraph $\mathcal{H}$ in Figure 3.1, $\mathcal{H}$ is partitioned into two clusters $\mathcal{C}_1$ and $\mathcal{C}_2$. We can observe that nodes v_1-v_5 in $\mathcal{C}_1$ share similar attributes and are closely connected to each other, whereas nodes v_6, v_7 and v_8 form a cluster $\mathcal{C}_2$ that is separated from $\mathcal{C}_1$ with a paucity of connections and distinct attributes.

3.3 Attributed Hypergraph Clustering

As mentioned, we first focus on attributed hypergraph clustering (AHC) and present our method `AHCKA` [73] in Sections 3.3, 3.4, and 3.5. Specifically, we will devise a random walk scheme on a K-nearest neighbor augmented hypergraph and present the AHC objective in Section 3.3, conduct theoretical analysis to support the design of `AHCKA` in Section 3.4, and develop the algorithmic details of `AHCKA` in Section 3.5.

For the problem of AHC, a central challenge is how to simultaneously exploit both hypergraph structure and attribute information for improved clustering quality. In

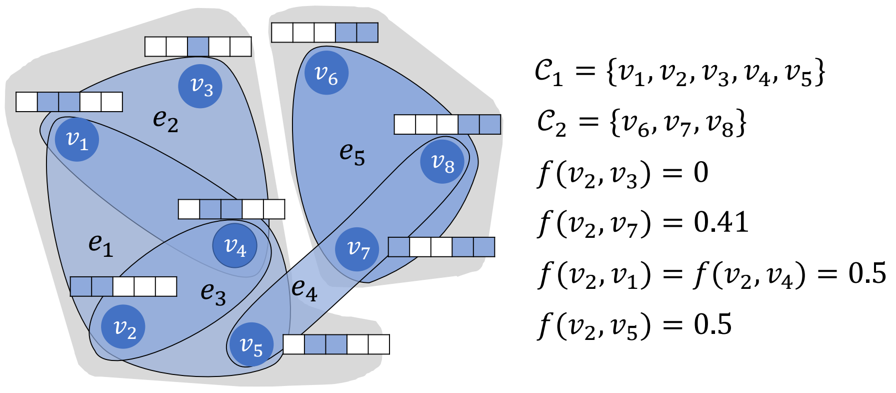


Figure 3.1: An Example Attributed Hypergraph

literature, it is a natural and effective approach to augment network structure with attribute similarity strengths [144, 13]. However, since a hypergraph yields different topological characteristics as illustrated in Figure 3.1, we argue that attribute augmentation should be conducted in a *controlable* way; otherwise, attributes may hamper, instead of improving, clustering quality, as shown in experiments (Section 3.8.3).

Therefore, in this section, we first develop a carefully-crafted augmentation strategy to augment attributes of nodes with hypergraph topology, which will benefit the clustering quality shown later on. As this augmentation strategy is orthogonal to the topological nature of hypergraph, its application to other types of networks, such as attributed graphs and attributed multiplex graphs, will be explained shortly in Section 3.6. Then we formulate Attributed Hypergraph Clustering as *Augmented Hypergraph Clustering*, with the same abbreviation AHC. The augmented hypergraph involves both hypergraph connections as well as augmented attribute connections. It is challenging to define a unified way to preserve the high-order information of both sides. To tackle this, we design the (α, β, γ)*-random walk* to uniformly model the node relationships (in terms of both the structural closeness and attribute similarity) in the augmented hypergraphs. Based thereon, we define a multi-hop conductance (MHC), and formulate the objective of AHC as optimizing the conductance.

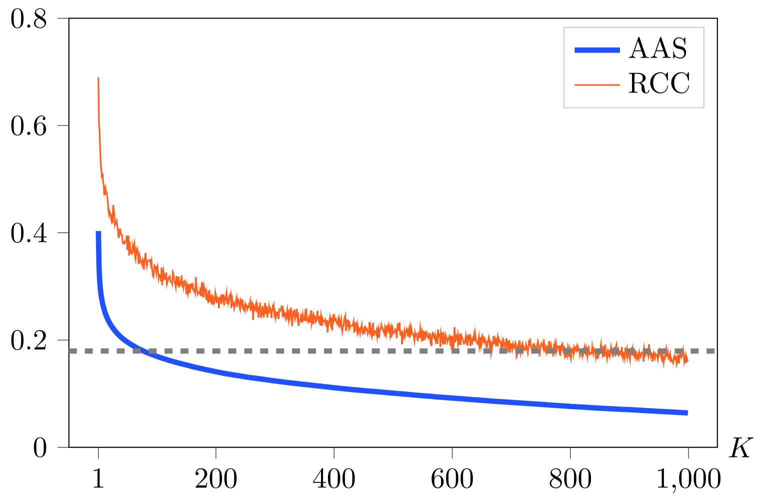


Figure 3.2: AAS and RCC on Cora-CA (best viewed in color)

3.3.1 KNN Augmentation

Although the vanilla augmentation strategy improves the clustering quality in attributed graphs [144, 13], to our knowledge, its effectiveness over attributed hypergraphs is as of yet under-explored. Moreover, it requires constructing a densely connected graph, causing severe efficiency issues on large graphs. To this end, we first demystify the attribute homogeneity of nodes within the same cluster through an empirical study on a real-world attributed hypergraph, i.e., the Cora-CA dataset[1] containing 2.7k academic papers in 7 research fields (i.e., 7 clusters). Every node has an attribute vector indicating the presence of words in the corresponding publication. First, we use $f(v_i, v_j) = cosine(\mathbf{X}[i], \mathbf{X}[j])$ to denote the attribute similarity of nodes v_i, v_j. We refer to v_j as the K-th nearest neighbor of v_i if $f(v_i, v_j)$ is the K-th largest $\forall v_j \in \mathcal{V} \setminus v_i$. Figure 3.2 plots the *averaged attribute similarity* (AAS for short) $f(v_i, v_j)$ of any randomly picked node v_i and its K-th nearest neighbor v_j, and their ratio of co-occurring in the same cluster (RCC for short), when varying K from 1 to 1000. The AAS and RCC results from this real-world example demonstrate that two nodes with higher attribute similarity are also more likely to appear in the same clus-

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

ter. Intuitively, applying the attribute-based augmentation strategy to hypergraphs can enhance the clustering results.

However, excessively augmenting the hypergraph with attribute information, namely, building too many connections between nodes according to attributes, will introduce distortion and adversely impact the clustering performance. To illustrate this, consider the example in Figure 3.1, where nodes v_2, v_3 are in the same cluster as they share multiple common neighbors while v_2, v_7 are not. If we were to assign a cluster to node v_2 as per the additional connections created by attribute similarities, it is more likely to be v_2, v_7 rather than v_2, v_3 in the same cluster given $f(v_2, v_7) = 0.41 > f(v_2, v_3) = 0$, which is counter-intuitive.

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

$$\mathbf{A}_K[i,j] = \begin{cases} 0, & \text{if } v_i \notin N_K(v_j) \text{ and } v_j \notin N_K(v_i), \\ 2 \cdot f(\mathbf{X}[i], \mathbf{X}[j]), & \text{if } v_i \in N_K(v_j) \text{ and } v_j \in N_K(v_i), \\ f(\mathbf{X}[i], \mathbf{X}[j]), & \text{otherwise.} \end{cases} \tag{3.1}$$

Thus, we obtain an augmented hypergraph $\mathcal{H}_A$ containing the hypergraph $\mathcal{H}_O = (\mathcal{V}, \mathcal{E})$ and the KNN graph $\mathcal{G}_K = (\mathcal{V}, \mathcal{E}_K)$. The reasons that we only consider K nearest neighbors for augmented hypergraph construction are three-fold. In the first place, the case study in Figure 3.2 suggests that there is no significant difference between the RCC of two random nodes (depicted by the gray dashed line) and that of two nodes v_i, v_j such that $v_j \in N_K(v_i)$, when K is beyond a number (roughly 500 in Figure 3.2). Therefore, such connections can be overlooked without impeding the clustering quality. Secondly, if we revisit the example in Figure 3.1 and apply the KNN strategy ($K = 3$) here, we can exclude the connection between v_2 and v_7 from $\mathcal{G}_K$ since $f(v_2, v_1) = f(v_2, v_4) = f(v_2, v_5) = 0.5 > f(v_2, v_7) = 0.41$. The distortion issue mentioned previously is therefore resolved. In comparison with the densely connected graph that encodes all attribute similarities (with up to $O(n^2)$ edges in the worst case), $\mathcal{G}_K$ can be efficiently constructed by utilizing well-established approximate nearest neighbor techniques with $O(n \log n)$ complexity [38, 50].

The range of the KNN neighborhood is determined by parameter K. While a larger K allows the KNN graph to include more attribute similarity relations, this also leads to a higher proportion of unwanted inter-cluster edges in the KNN graph as evidenced by the lower RCC in Figure 3.2. Meanwhile, K cannot be too small (e.g., 5), or it will fail to utilize highly similar nodes that usually have high RCC. The trade-off of choosing K is evaluated in Section 3.8.3.

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

3.3.2 (α, β, γ)-Random Walk

Random walk with restart [116] (RWR) is one of the most common and effective random walk models for capturing the multi-hop relationships between nodes in a graph [51], and is widely used in many tasks such as ranking [116, 108], recommendation [93], and clustering [1]. Given a graph $\mathcal{G}$, a source node u and a stopping probability

α (typically $\alpha = 0.2$), at each step, an RWR originating from u either stops at the current node with probability α, or randomly picks an out-neighbor v of the current node according to the weight of edge (u, v) and navigates to v with the remaining $1 - \alpha$ probability. It follows that RWR score (a.k.a. *personalized PageRank* [48]) of any node pair (u, v) represents the probability that an RWR from u ends at node v. Intuitively, two nodes with dense (one-hop or multi-hop) connections should have a high RWR score.

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

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

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

Each node v_i is associated with a parameter β_i (see Eq. (3.2)) used to control the joint navigation between hypergraph $\mathcal{H}_O$ and KNN $\mathcal{G}_K$. The larger β_i is, the more likely that the random walk jumps to the neighbors of v_i in KNN $\mathcal{G}_K$.

$$\beta_i = \begin{cases} 0, & \text{if } \mathbf{X}[i] \text{ is a zero vector;} \\ 1, & \text{else if } \delta(v_i) = 0; \\ \beta, & \text{otherwise.} \end{cases} \tag{3.2}$$

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

$$s(v_i, v_j) = \mathbf{S}[i, j] = \alpha \sum_{\ell=0}^{\gamma} (1 - \alpha)^{\ell} \mathbf{P}^{\ell}[i, j], \tag{3.3}$$

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

$$\mathbf{P} = (\mathbf{I} - \mathbf{B}) \cdot \mathbf{D}_V^{-1} \mathbf{H}^{\top} \mathbf{D}_E^{-1} \mathbf{H} + \mathbf{B} \mathbf{D}_K^{-1} \mathbf{A}_K, \tag{3.4}$$

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

3.3.3 Objective Function

In what follows, we formally define the objective function of AHC. Intuitively, a high-quality cluster $\mathcal{C}$ in the augmented hypergraph $\mathcal{H}_A$ should be both internally cohesive and well disconnected from the remainder of the graph with the consideration of multi-hop connections. Hence, if we simulate an (α, β, γ)-random walk W from any node in $\mathcal{C}$, W should have a low probability of escaping from $\mathcal{C}$, i.e., ending at any node outside $\mathcal{C}$. We refer to this escaping probability $\phi(\mathcal{C})$ as the *multi-hop conductance* (MHC) of $\mathcal{C}$, defined in Eq. (3.5).

$$\phi(\mathcal{C}) = \tfrac{1}{|\mathcal{C}|} \textstyle\sum_{v_i\in\mathcal{C}} \sum_{v_j\notin\mathcal{C}} s(v_i, v_j) \tag{3.5}$$

Since a low MHC $\phi(\mathcal{C})$ reflects a high coherence of cluster $\mathcal{C}$, we then formulate AHC as an optimization problem of finding k clusters $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ such that their MHC $\Phi(\{\mathcal{C}_1, \ldots, \mathcal{C}_k\})$ (Eq. (3.6)) is minimized.

$$\Phi(\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}) = \frac{1}{k} \sum_{\mathcal{C}\in\{\mathcal{C}_1,\ldots,\mathcal{C}_k\}} \frac{1}{|\mathcal{C}|} \sum_{v_i\in\mathcal{C}} \sum_{v_j\notin\mathcal{C}} s(v_i, v_j) \tag{3.6}$$

Directly minimizing Eq. (3.6) requires computing $s(v_i, v_j)$ (Eq. (3.3)) of every two nodes $v_i \in \mathcal{C}$, $v_j \in \mathcal{V}\setminus\mathcal{C}$, $\forall\mathcal{C} \in \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k\}$, which is prohibitively expensive due to intractable computation time (i.e., $O(n^3)$) and storage space (i.e., $O(n^2)$). In addition, the minimization of $\Phi(\{\mathcal{C}_1, \ldots, \mathcal{C}_k\})$ is an NP-complete combinatorial optimization problem [106], rendering the exact solution unattainable on large graphs.

3.4 Theoretical Analysis for `AHCKA`

This section presents the top-level idea of our proposed solution, `AHCKA`, to AHC computation, and explains the intuitions behind it. At a high level, `AHCKA` first transforms the objective of AHC in Eq. (3.6) to a matrix trace maximization problem, and then derives an approximate solution via a top-k eigendecomposition. Note that for any k non-overlapping clusters $\{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k\}$ on $\mathcal{H}$ satisfying $\bigcup_{i=1}^{k} \mathcal{C}_i = \mathcal{V}$, they can be represented by a binary matrix $\mathbf{Y} \in \{0, 1\}^{n\times k}$, where for each node v_i and cluster $\mathcal{C}_j$

$$\mathbf{Y}[i,j] = \begin{cases} 1, & v_i \in \mathcal{C}_j \\ 0, & v_i \in \mathcal{V} \setminus \mathcal{C}_j. \end{cases} \tag{3.7}$$

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

$$h(\mathbf{Y}) = (\mathbf{Y}^{\mathsf{T}}\mathbf{Y})^{-1/2}\mathbf{Y} = \widehat{\mathbf{Y}} \tag{3.8}$$

to stand for the L_2 normalization of $\mathbf{Y}$. Particularly, $\widehat{\mathbf{Y}}$ has orthonormal columns, i.e., $\widehat{\mathbf{Y}}^{\mathsf{T}}\widehat{\mathbf{Y}} = \mathbf{I}_k$ where $\mathbf{I}_k$ is a $k \times k$ identity matrix. Given k non-overlapping clusters $\{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k\}$ and their corresponding BCM matrix $\mathbf{Y}$, it is trivial to show

$$\Phi(\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}) = 1 - \Psi(\mathbf{Y}), \tag{3.9}$$

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

$$\Psi(\mathbf{Y}) = \frac{1}{k} trace(\widehat{\mathbf{Y}}^{\mathsf{T}}\mathbf{S}\widehat{\mathbf{Y}}). \tag{3.10}$$

Eq. (3.9) suggests that the minimization of MHC $\Phi(\{\mathcal{C}_1, \ldots, \mathcal{C}_k\})$ is equivalent to finding a BCM matrix $\mathbf{Y}$ such that the trace of matrix $\widehat{\mathbf{Y}}^{\mathsf{T}}\mathbf{S}\widehat{\mathbf{Y}}$ is maximized. Due to its NP-completeness, instead of computing the exact solution, we utilize a two-phase strategy to derive an approximate solution as follows.

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

Lemma 3.4.1. *Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$ be the k largest singular values of matrix $\mathbf{S}$ in Eq. (3.3). Given any matrix $\mathbf{W} \in \mathbb{R}^{n \times k}$ such that $h(\mathbf{W})$ satisfies $h(\mathbf{W})^{\top} \cdot h(\mathbf{W}) = \mathbf{I}_K$, then $\Psi(\mathbf{W}) \leq \frac{1}{k}\sum_{i=1}^{k} \sigma_i = \psi_\sigma$.*

Proof. Let $\widehat{\mathbf{W}}$ denote $h(\mathbf{W})$, i.e., $(\mathbf{W}^\intercal\mathbf{W})^{-1/2}\mathbf{W}$ (Eq. (3.8)) , and we have

$$\Psi(\mathbf{W}) = \frac{1}{k}trace(\widehat{\mathbf{W}}^\intercal\mathbf{S}\widehat{\mathbf{W}}) = \frac{1}{k}trace(\mathbf{S}(\widehat{\mathbf{W}}\widehat{\mathbf{W}}^\intercal)) \tag{3.11}$$

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

$$\begin{aligned} k &= rank(\widehat{\mathbf{W}}) + rank(\widehat{\mathbf{W}}^\intercal) - k \\ &\leq rank(\widehat{\mathbf{W}}\widehat{\mathbf{W}}^\intercal) \leq \min(rank(\widehat{\mathbf{W}}), rank(\widehat{\mathbf{W}}^\intercal)) = k \end{aligned} \tag{3.12}$$

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

$$\Psi(\mathbf{W}) \leq \frac{1}{k}(\sum_{i=1}^{k} 1 \cdot \sigma_i + \sum_{i=k+1}^{n} 0 \cdot \sigma_i) = \frac{1}{k}\sum_{i=1}^{k} 1 \cdot \sigma_i = \psi_\sigma \tag{3.13}$$

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

Lemma 3.4.1 implies that if we can first find a fractional matrix $\mathbf{W}$ such that $\Psi(\mathbf{W})$ is close to ψ_σ, a high-quality BCM matrix $\mathbf{Y}$ can be converted from $\mathbf{W}$ by leveraging algorithms such as k-`Means` [80]. Although we can obtain such a fractional matrix $\mathbf{W}$ by applying trace maximization techniques [124] to Eq. (3.10), it still remains tenaciously challenging to compute $\mathbf{S}$.

Lemma 3.4.2. *Let the columns of $\mathbf{Q} \in \mathbb{R}^{n\times k}$ be the second to $(k+1)$-th leading eigenvectors of $\mathbf{P}$ (Eq. (3.4)). Then, we have $\Psi(\mathbf{Q}) = \frac{1}{k}\sum_{i=2}^{k+1}\lambda_i = \psi_\lambda$, where $\lambda_2 \geq \cdots \geq \lambda_k \geq \lambda_{k+1}$ are the second to $(k+1)$-th leading eigenvalues of $\mathbf{S}$, sorted by algebraic value in descending order.*

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

$$\mathbf{S}e_i = (\alpha \textstyle\sum_{l=0}^{\gamma}(1-\alpha)^l\mathbf{P}^l)e_i = (\alpha \sum_{l=0}^{\gamma}(1-\alpha)^l\lambda_i'^l)e_i \tag{3.14}$$

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

We exclude the first eigenvector $\frac{1}{\sqrt{n}}\cdot\mathbf{1}$ of $\mathbf{P}$ as it is useless for clustering. By virtue of our analysis in Lemma 3.4.2, the second to $(k+1)$-th leading eigenvectors $\mathbf{Q}$ of $\mathbf{P}$ (see Eq. (3.4)) can be regarded as a rough $\mathbf{W}$ since $\Psi(\mathbf{Q}) = \psi_\lambda \leq \psi_\sigma$ and the gap between ψ_λ and ψ_σ is insignificant in practice. For instance, on the Cora-CA dataset, we can obtain $\psi_\sigma = 0.668$ and $\psi_\lambda = 0.596$ (*i.e.*, $\Phi_\sigma = 1-\psi_\sigma = 0.332$, $\Phi_\lambda = 1-\psi_\lambda = 0.404$), both of which are better than $\Psi(\mathbf{Y}^*) = 0.533$ (*i.e.*, $\Phi^* = 1-\Psi(\mathbf{Y}^*) = 0.467$) of the ground-truth BCM matrix $\mathbf{Y}^*$. Consequently, using the second to $(k+1)$-th leading eigenvectors $\mathbf{Q}$ of $\mathbf{P}$ as the fractional solution $\mathbf{W}$ is sufficient to derive a favorable BCM matrix. Moreover, in doing so, we can avoid the tremendous overhead incurred by the materialization of $\mathbf{S}$.

To summarize, AHCKA adopts a two-phase strategy to obtain an approximate solution

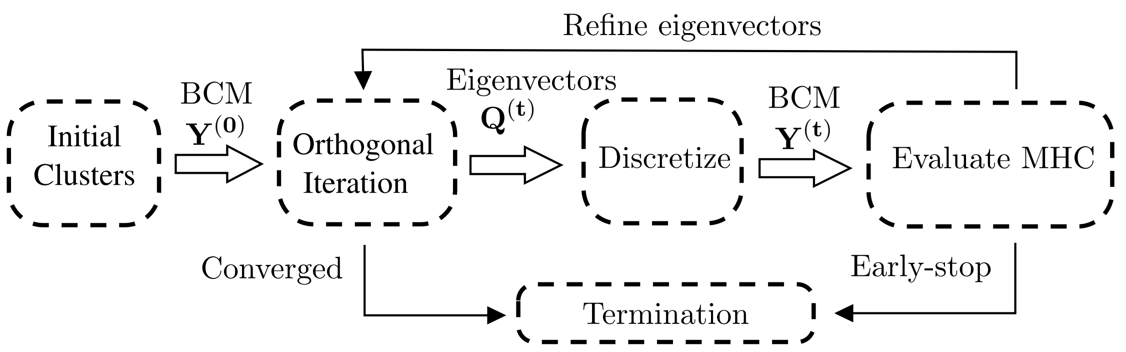


Figure 3.3: Overview of `AHCKA`

to the AHC problem. First, `AHCKA` computes the second to $(k+1)$-th leading eigenvectors $\mathbf{Q}$ of $\mathbf{P}$. After that, `AHCKA` transforms $\mathbf{Q}$ into a BCM matrix $\mathbf{Y}$ through a discretization approach [148] that minimizes the difference between $\mathbf{Q}$ and $\mathbf{Y}$. The rationale is that $\Psi(\mathbf{Q}) = \Psi(\mathbf{QR})$ if $\mathbf{R}$ is a $k \times k$ orthogonal matrix, ensuring $\mathbf{R}^{\mathsf{T}}\mathbf{R} = \mathbf{I}_k$. Accordingly, we can derive a BCM matrix $\mathbf{Y} = \mathbf{QR}$ by minimizing the Frobenius norm $||\mathbf{Q} - \mathbf{QR}||_F$ with a binary constraint exerted on $\mathbf{QR}$. Note that we do not adopt k-`Means` over $\mathbf{Q}$ to get the BCM matrix $\mathbf{Y}$ as it deviates from the objective in Eq. (3.10), and thus, produces sub-par result quality, as revealed by experiments (Table 3.14).

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

1. The brute-force computation of $\mathbf{Q}$ is time-consuming as it requires numerous iterations and the construction of $\mathbf{P}$.

2. In practice, directly utilizing the exact or near-exact $\mathbf{Q}$ might incur overfitting towards the objective instead of ground-truth clusters, and hence, lead to sub-optimal clustering quality. It is challenging to derive a practically effective and robust BCM matrix $\mathbf{Y}$ from $\mathbf{Q}$.

3.5 The AHCKA Algorithm

To circumvent the above challenges, AHCKA integrates the aforementioned two-phase scheme into an iterative framework, which enables us to approximate the second to $(k+1)$-th leading eigenvectors $\mathbf{Q}$ without constructing $\mathbf{P}$ explicitly, and greedily search the BCM matrix $\mathbf{Y}$ with the best MHC. Figure 3.3 sketches the main ingredients and algorithmic procedure of AHCKA. More specifically, AHCKA employs *orthogonal iterations* [104] to approximate the second to $(k+1)$-th leading eigenvectors $\mathbf{Q}$ of $\mathbf{P}$. During the course, AHCKA starts with an initial BCM matrix, followed by an orthogonal iteration to compute an approximate $\mathbf{Q}$ and an updated BCM matrix $\mathbf{Y}$ from the $\mathbf{Q}$ through Discretize algorithm [148]. Afterward, AHCKA inspects if $\mathbf{Q}$ reaches convergence and computes the MHC with the current BCM matrix $\mathbf{Y}$ via CalMHC algorithm (Algorithm 2). If $\mathbf{Q}$ converges (i.e., the BCM remains nearly stationary) or the early termination condition is satisfied (i.e., the MHC of current $\mathbf{Y}$ is satisfying), AHCKA terminates. Otherwise, AHCKA enters into the next orthogonal iteration with the updated $\mathbf{Q}$ and $\mathbf{Y}$.

In what follows, a detailed description of AHCKA is given in Section 3.5.1. Section 3.5.2 introduces an effective approach InitBCM for initializing the BCM matrix $\mathbf{Y}$, which drastically curtails the number of iterations needed and significantly boosts the computation efficiency of AHCKA. The complexity of the complete algorithm is analyzed in Section 3.5.3.

3.5.1 Main Algorithm

The pseudo-code of AHCKA is presented in Algorithm 1, which takes as input an attributed hypergraph $\mathcal{H}$, transition matrix of attribute KNN graph $\mathbf{P}_K$, the number k of clusters, a diagonal matrix $\mathbf{B}$ containing n parameters defined in Eq. (3.2), the random walk stopping probability α, an error threshold ϵ_Q, the numbers γ, T_a of

Algorithm 1: AHCKA

```
Input: Hypergraph $\mathcal{H}$, KNN transition matrix $\mathbf{P}_K$, the number of clusters $k$,
       diagonal matrix $\mathbf{B}$, constant $\alpha$, error threshold $\epsilon_Q$, the numbers of
       iterations $T_a$, $\gamma$, an integer $\tau$, and an initial BCM matrix $\mathbf{Y}^{(0)}$.
Output: BCM matrix $\mathbf{Y}$
1  $\mathbf{Y} \leftarrow \mathbf{Y}^{(0)}$, $\widehat{\mathbf{Y}}^{(0)} \leftarrow h(\mathbf{Y}^{(0)})$;
2  $\mathbf{Q}^{(0)} \leftarrow \frac{1}{\sqrt{n}} \cdot \mathbf{1} | \widehat{\mathbf{Y}}^{(0)}$ ;
3  for $t \leftarrow 1, 2, \cdots, T_a$ do
4      Compute $\mathbf{Z}^{(t)}$ according to Eq. (3.16);
5      $\mathbf{Q}^{(t)}, \mathbf{R}^{(t)} \leftarrow$ QR($\mathbf{Z}^{(t)}$) ;
6      if $t$ mod $\tau = 0$ then
7          $\mathbf{Y}^{(t)} \leftarrow$ Discretize($\mathbf{Q}^{(t)}$) ;
8          $\Phi(\mathbf{Y}^{(t)}) \leftarrow$ CalMHC($\mathbf{Y}^{(t)}, \mathbf{P}_V, \mathbf{P}_E, \mathbf{P}_K, \mathbf{B}, \gamma, \alpha$);
9          if $\Phi(\mathbf{Y}^{(t)}) < \Phi(\mathbf{Y})$ then $\mathbf{Y} \leftarrow \mathbf{Y}^{(t)}$;
10         if Eq. (3.19) or Eq. (3.20) holds then break;
11 return $\mathbf{Y}$;
```

iterations, an integer τ, and an initial BCM matrix $\mathbf{Y}^{(0)}$. AHCKA starts by computing the normalized BCM matrix $\widehat{\mathbf{Y}}^{(0)} = h(\mathbf{Y}^{(0)})$ (Eq. (3.8)) and setting the initial $k+1$ leading eigenvectors $\mathbf{Q}^{(0)}$ as $\frac{1}{\sqrt{n}} \cdot \mathbf{1} | \widehat{\mathbf{Y}}^{(0)}$ (Lines 1-2), where $|$ represents the horizontal concatenation and $\frac{1}{\sqrt{n}} \cdot \mathbf{1}$ is the first leading eigenvector of $\mathbf{P}$ since it is a stochastic matrix. After that, AHCKA enters into at most T_a orthogonal iterations for computing the $k+1$ leading eigenvectors $\mathbf{Q}$ and the BCM matrix $\mathbf{Y}$ (Lines 3-10). At step t, orthogonal iteration updates the approximate $k+1$ leading eigenvectors of $\mathbf{P}$ as $\mathbf{Q}^{(t)}$ by the formula below (Lines 4-5):

$$\mathbf{Q}^{(t)}\mathbf{R}^{(t)} = \mathbf{Z}^{(t)} = \mathbf{P}\mathbf{Q}^{(t-1)}, \tag{3.15}$$

where $\mathbf{Q}^{(t)}$ is obtained by a QR decomposition over $\mathbf{Z}^{(t)}$. If t is sufficiently large, $\mathbf{Q}^{(t)}$ will converge to the exact $k+1$ leading eigenvectors of $\mathbf{P}$ [104]. Note that the direct computation of $\mathbf{Z}^{(t)} = \mathbf{P}\mathbf{Q}^{t-1}$ requires constructing $\mathbf{P}$ explicitly as per Eq. (3.4), which incurs an exorbitant amount of time and space (up to $O(n^2)$ in the worst case).

Algorithm 2: `CalMHC`

```
Input: $\mathbf{Y}^{(t)}, \mathbf{P}_V, \mathbf{P}_E, \mathbf{P}_K, \mathbf{B}, \gamma, \alpha$
Output: MHC $\phi_t$
1 $\widehat{\mathbf{Y}}^{(t)} \leftarrow h(\mathbf{Y}^{(t)})$; $\mathbf{F}^{(0)} \leftarrow \alpha\widehat{\mathbf{Y}}^{(t)}$;
2 for $\ell \leftarrow 1, 2, \ldots \gamma$ do
3     Compute $\mathbf{F}^{(\ell)}$ according to Eq. (3.18);
4 $\phi_t \leftarrow 1 - \frac{1}{k} trace(\widehat{\mathbf{Y}}^{(t)\top}\mathbf{F}^{(\gamma)})$;
5 return $\phi_t$ ;
```

To mitigate this, we decouple and reorder the matrix multiplication as in Eq. (3.16).

$$\mathbf{Z}^{(t)} = (\mathbf{I} - \mathbf{B}) \cdot \mathbf{P}_V \cdot \left(\mathbf{P}_E \mathbf{Q}^{(t-1)}\right) + \mathbf{B}\mathbf{P}_K \cdot \mathbf{Q}^{(t-1)}, \tag{3.16}$$

$$\text{where } \mathbf{P}_V = \mathbf{D}_V^{-1}\mathbf{H}^\top, \;\; \mathbf{P}_E = \mathbf{D}_E^{-1}\mathbf{H} \tag{3.17}$$

$\mathbf{P}_V$ and $\mathbf{P}_E$ are two sparse matrices of $\mathcal{H}$ and $\mathbf{P}_K = \mathbf{D}_K^{-1}\mathbf{A}_K$ is the sparse transition matrix of the KNN graph $\mathcal{G}_K$ defined in Section 3.3.1. Note that all of them can be efficiently constructed in the preprocessing stage. As such, we eliminate the need to materialize $\mathbf{P}$ and reduce the time complexity of computing $\mathbf{Z}^{(t)}$ to $O(nk \cdot (\overline{\delta} + K))$.

After obtaining $\mathbf{Q}^{(t)}$, `AHCKA` converts $\mathbf{Q}^{(t)}$ into a new BCM matrix $\mathbf{Y}^{(t)}$ (Lines 6-7) using the `Discretize` algorithm [148]. Notice that we conduct this conversion every other τ iterations in order to avert unnecessary operations as the difference between $\mathbf{Y}^{(t)}$ and $\mathbf{Y}^{(t-1)}$ is often insignificant.

Next, at Line 8, `AHCKA` invokes `CalMHC` (i.e., Algorithm 2) with a BCM matrix $\mathbf{Y}^{(t)}$, other parameters including $\mathbf{P}_V$, $\mathbf{P}_E$, $\mathbf{P}_K$, $\mathbf{B}$, α, and the number of iterations γ as input to calculate the MHC ϕ_t of the current BCM matrix $\mathbf{Y}^{(t)}$. To avoid the materialization of $\mathbf{S}$ required in Eq. (3.9) and Eq. (3.10), Algorithm 2 computes ϕ_t in an iterative manner by reordering the matrix multiplications (Lines 2-3 in Algorithm 2). More precisely, at the ℓ-th iteration, it obtains the intermediate result $\mathbf{F}^{(\ell)}$ via the following equation:

$$\mathbf{F}^{(\ell)} = (1-\alpha)\left((\mathbf{I}-\mathbf{B})\cdot\mathbf{P}_V\cdot\left(\mathbf{P}_E\mathbf{F}^{(\ell-1)}\right)+\mathbf{B}\mathbf{P}_K\cdot\mathbf{F}^{(\ell-1)}\right)+\mathbf{F}^{(0)}. \tag{3.18}$$

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

Once the convergence criterion of $\mathbf{Q}^{(t)}$ (Eq. (3.19)) is satisfied, or the early termination condition (Eq. (3.20)) holds, `AHCKA` ceases the iterative process and returns the BCM matrix $\mathbf{Y}$ with the lowest MHC (Lines 9-11 in Algorithm 1).

$$||\mathbf{Q}^{(t)} - \mathbf{Q}^{(t-1)}|| < \epsilon_Q \tag{3.19}$$

$$\phi_{t-2\tau} < \phi_{t-\tau} < \phi_t \tag{3.20}$$

Otherwise, `AHCKA` proceeds to the next orthogonal iteration. The rationale for the early termination condition in Eq. (3.20) is that, in practice, successive increases in ϕ_t indicate that clusters with desirable MHC objective have been attained.

3.5.2 Greedy Initialization of BCM

Akin to many optimization problems, `AHCKA` requires many iterations to achieve convergence when $\mathbf{Y}^{(0)}$ is randomly initialized. To tackle this issue, we propose a greedy initialization technique, `InitBCM`, whereby we can immediately gain a passable BCM matrix $\mathbf{Y}^{(0)}$ and expedite the convergence, as demonstrated by our experiments in Section 3.8.4.

The rationale of `InitBCM` is that most nodes tend to cluster together around a number of center nodes [101]. Therefore, we can first pick a set $\mathcal{V}_c$ of top influential nodes w.r.t. the whole hypergraph, and calculate the multi-hop proximities (i.e., RWR scores) of each node to the influential nodes $\mathcal{V}_c$ (i.e., centers). Then, the cluster center of each

Algorithm 3: InitBCM

```
Input: Hypergraph $\mathcal{H}$, matrices $\mathbf{P}_V, \mathbf{P}_E$, integer $k$, constant $\alpha$, the number of
       iterations $T_i$.
Output: An initial BCM matrix $\mathbf{Y}^{(0)}$.
1  $\mathcal{V}_c \leftarrow$ The sorted indices of nodes with $k$ largest degrees;
2  Initialize $\mathbf{Z}_0 \leftarrow \mathbf{0}^{k \times n}$;
3  for $j \leftarrow 1$ to $k$ do $\mathbf{Z}_0[j, \mathcal{V}_c[j]] \leftarrow 1$ ;
4  Initialize $\mathbf{\Pi}_c^{(0)} \leftarrow \alpha \mathbf{Z}_0$;
5  for $t \leftarrow 1, 2, \ldots T_i$ do
6      Compute $\mathbf{\Pi}_c^{(t)}$ according to Eq. (3.21);
7  for $v_j \in \mathcal{V}$ do
8      Calculate $g(v_j)$ according to Eq. (3.22);
9      $\mathbf{Y}^{(0)}[j, g(v_j)] \leftarrow 1$;
10 return $\mathbf{Y}^{(0)}$ ;
```

node can be determined by its proximity to nodes in $\mathcal{V}_c$ accordingly.

Algorithm 3 displays the pseudo-code of InitBCM. Given hypergraph $\mathcal{H}$, and transition matrices $\mathbf{P}_V, \mathbf{P}_E$ defined in Eq. (3.17), the number k of clusters, random walk stopping probability α, and the number of iterations T_i, as input, InitBCM begins by initializing an ordered set $\mathcal{V}_c$ consisting of the k nodes with k largest degrees in $\mathcal{H}$ (sorted by their indices), which later serves as the cluster centers (Line 1). Then, a $k \times n$ matrix $\mathbf{Z}_0$ is created, where for each integer $j \in [1, k]$, $\mathbf{Z}_0[j, \mathcal{V}_c[j]]$ is set to 1 and 0 otherwise and $\mathcal{V}_c[j]$ denotes the node index of the j-th node in $\mathcal{V}_c$ (Lines 2-3). Next, InitBCM launches T_i iterations to calculate the RWR scores of all nodes w.r.t the k nodes in $\mathcal{V}_c$ (Lines 5-6). Specifically, at t-th iteration, we compute approximate RWR $\mathbf{\Pi}_c^{(t)}$ (Line 6):

$$\mathbf{\Pi}_c^{(t)} = (1 - \alpha) \left(\mathbf{\Pi}_c^{(t-1)} \mathbf{P}_V \right) \cdot \mathbf{P}_E + \mathbf{\Pi}_0, \tag{3.21}$$

where $\mathbf{\Pi}_0 = \alpha \mathbf{Z}_0$ (Line 4). Note that we reorder the matrix multiplications as in Eq. (3.21) so as to bypass the materialization of the $n \times n$ matrix $\mathbf{P}_V \mathbf{P}_E$. After obtaining $\mathbf{\Pi}_c^{(T_i)}$, InitBCM assigns the node $\mathcal{V}_c[g(v_j)]$ as the cluster center to each node v_j in $\mathcal{H}$

as per Eq. (3.22) (Lines 7-9).

$$g(v_j) = \arg\max_{1\le l\le k} \mathbf{\Pi}_c^{(T_i)}[l,j], \tag{3.22}$$

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

3.5.3 Complexity

One of the main computational costs of `AHCKA` stems from the sparse matrix multiplications, i.e., Line 4 in Algorithm 1, Line 3 in Algorithm 2, and Line 6 in Algorithm 3. We first consider Line 4 in Algorithm 1, i.e., Eq. (3.16). Since $\mathbf{Q}^{(t-1)}$ is an $n\times(k+1)$ matrix and the numbers of non-zero entries in sparse matrices $\mathbf{P}_V$, $\mathbf{P}_E$, and $\mathbf{P}_K$ are $n\overline{\delta}$, $n\overline{\delta}$, and nK, respectively, its complexity is $O((n\overline{\delta}+nK)\cdot k)$ [150]. Analogously, according to Eq. (3.18), and Eq. (3.21), both the time costs of Line 3 in Algorithm 2 and Line 6 in Algorithm 3 are bounded by $O(n\overline{\delta}k)$. Recall that these three operations are conducted up to T_a, γ, and T_i times in Algorithms 1, 2, and 3, respectively. Therefore, the total time cost of sparse matrix multiplications is $O(kn\overline{\delta}\cdot(T_a+T_i+\gamma)+knKT_a)$. Moreover, in Algorithm 1, the `QR` decomposition at Line 5 takes $O(k^2n)$ time and `Discretize` [148] runs in $O(k^2n+k^3)$ time. Overall, the time complexity of `AHCKA` is $O(kn\overline{\delta}\cdot(T_a+T_i+\gamma)+knKT_a+k^2n)$, which equals $O(n\overline{\delta})$ when T_a, T_i, γ, k, and K are regarded as constants. The space complexity of `AHCKA` is $O(n\cdot(\overline{\delta}+K+k))$ as all matrices are in sparse form.

3.6 The ANCKA framework

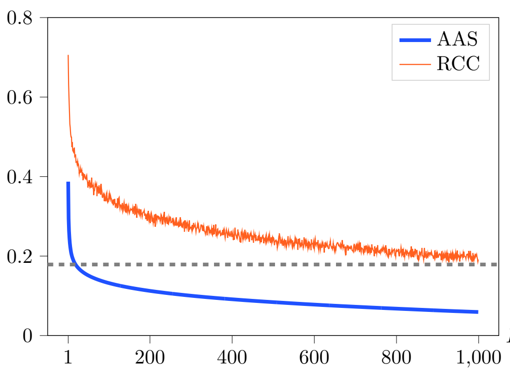
In this section, we generalize AHCKA that is for AHC to a versatile framework ANCKA to process all of AHC, AGC, and AMGC, formulated in Section 3.2. ANCKA aims to efficiently find high-quality clusters on various types of network $\mathcal{N}$.

As mentioned, the proposed KNN augmentation in Section 3.3.1 is orthogonal to the high-order nature of hypergraph, and therefore, we can apply the KNN augmentation to input attributed network $\mathcal{N}$ that can be an attributed hypergraph $\mathcal{H}$, graph $\mathcal{G}$, and multiplex graph $\mathcal{G}_M$.

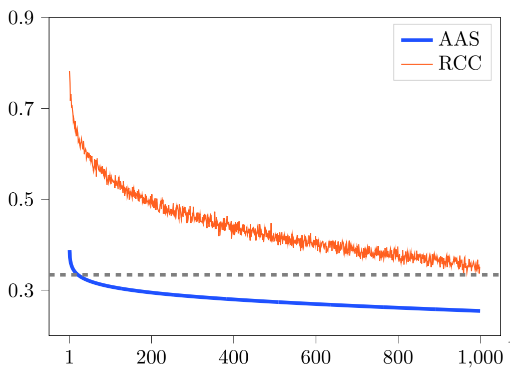
Recall that, in Figure 3.2, we have empirically shown that nodes with higher attribute similarity are more likely to appear in the same cluster of a hypergraph $\mathcal{H}$. This also holds for attributed graphs and attributed multiplex graphs. Figures 3.4a-3.4b illustrate the AAS and RCC on the attributed graph Citeseer-DG and the attributed multiplex graph ACM, with binary keyword vectors as node attributes. On both datasets, nodes with higher attribute similarity (i.e., higher AAS with smaller K) are more likely to be in the same cluster (i.e., higher RCC). Moreover, above a certain K value, there is no significant difference between the RCC of two random nodes and that of two nodes v_i and v_j such that v_j is the K-nearest neighbor of v_i. Based on these observations, it is viable to extend KNN augmentation in Section 3.3.1 to an attributed network $\mathcal{N}$ with n nodes and attribute matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, by building a KNN augmentation graph $\mathcal{G}_K$ via Eq. (3.1).

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

3.6.1 Generalized (α, β, γ)-Random Walk



(a) AAS and RCC on Citeseer-DG



(b) AAS and RCC on ACM

For the augmented network $\mathcal{N}_A = (\mathcal{N}_O, \mathcal{G}_K)$, define $\mathbf{P}_N$ and $\mathbf{P}_K$ as the random walk transition matrices of $\mathcal{N}_O$ and $\mathcal{G}_K$ respectively. The generalized (α, β, γ)-random walk on $\mathcal{N}_A$ is an RWR process over the augmented network $\mathcal{N}_A$, similar to the case of attributed hypergraphs in AHCKA. The difference from Definition 1 is that when the random walk navigates to another node, with probability $1 - \beta_i$, an out-neighbor is drawn from the distribution of $\mathbf{P}_N$ instead of incident hyperedges. This generalized random walk can also be characterized by the probability in Eq. (3.3), with transition matrix $\mathbf{P}$ given as follows.

$$\mathbf{P} = (\mathbf{I} - \mathbf{B}) \cdot \mathbf{P}_N + \mathbf{B} \cdot \mathbf{P}_K. \tag{3.23}$$

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

Attributed Hypergraph $\mathcal{H}$. When $\mathcal{N}_O$ is a hypergraph with hyperedge incidence matrix $\mathbf{H}$, based on Eq. (3.4), $\mathbf{P}_N$ is shown below. $\mathbf{P}_N$ considers the transition probability $\mathbf{P}_V$ from a node to its incident hyperedges and the transition probability $\mathbf{P}_E$ from each hyperedge to nodes connected by the hyperedge.

$$\mathbf{P}_N = \mathbf{P}_V \mathbf{P}_E, \text{ where } \mathbf{P}_V = \mathbf{D}_V^{-1}\mathbf{H}^\top \text{ and } \mathbf{P}_E = \mathbf{D}_E^{-1}\mathbf{H}. \tag{3.24}$$

Attributed Graph $\mathcal{G}$. When $\mathcal{N}_O$ is an undirected graph, we can acquire the transition matrix $\mathbf{P}_N$ in Eq. (3.25). If $\mathcal{N}_O$ is directed, we introduce a reversed edge for each edge and consider bidirectional connections between nodes to get $\mathbf{A}$, $\mathbf{D}$, and subsequently $\mathbf{P}_N$.

$$\mathbf{P}_N = \mathbf{D}^{-1}\mathbf{A}, \tag{3.25}$$

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

Attributed Multiplex Graph $\mathcal{G}_M$. When $\mathcal{N}_O$ is a multiplex graph comprising L layers with the same node set $\mathcal{V}$, the l-th layer has its own edge set $\mathcal{E}_l$ representing a unique type of connections. The overall goal of the clustering task is to make cluster assignments that capture the collective structure of the multiplex graph, transcending the differences across layers. To achieve this, intuitively, we treat every layer equally and compute $\mathbf{P}_N$ as in Eq. (3.26), while layer weighting is left as future work [52]. Given the degree matrix $\mathbf{D}_l$ and adjacency matrix $\mathbf{A}_l$ of every l-th layer, we get the layer's random walk transition matrix $\mathbf{D}_l^{-1}\mathbf{A}_l$, and then compute $\mathbf{P}_N$ of the multiplex graph by averaging the layer-specific transition matrices. Consequently, from the current node v, a random walk has $1/L$ probability of selecting each layer $\mathcal{G}_l$, and then within this chosen layer, the next node to visit is picked uniformly at random from the out-neighbors of v in $\mathcal{G}_l$.

$$\mathbf{P}_N = \frac{1}{L}\sum_{l=1}^{L}\mathbf{D}_l^{-1}\mathbf{A}_l, \tag{3.26}$$

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

3.6.2 ANCKA Algorithm

With the random walk transition matrix $\mathbf{P}$ formulated above for various types of attributed networks $\mathcal{N}$, Eq. (3.3) can be reused to calculate $\mathbf{S}[i,j]$, the probability of a generalized (α,β,γ)-random walk from v_i stopping at v_j in the end. The objective function in Section 3.3.3 is naturally extended to `ANCKA`. Consequently, our theoretical analysis in Section 3.4 remains valid for `ANCKA` over attributed networks that can be hypergraphs, graphs, and multiplex graphs.

The pseudo-code of `ANCKA` is outlined in Algorithm 4. At Line 1, it obtains transition matrix $\mathbf{P}_K$ for attribute KNN augmentation. Then as a framework supporting various attributed networks, `ANCKA` is a generalization of Algorithms 1-3 with transition matrix $\mathbf{P}_N$ computed depending on the network type at Line 2. $\mathbf{P}_N$ is then used throughout the algorithm as a part of the generalized (α, β, γ)-random walk. The greedy initialization of clusters in Lines 3-11 resembles the procedure in `InitBCM` with the corresponding $\mathbf{P}_N$ for RWR simulation. Since `ANCKA` needs to pick k nodes in $\mathcal{N}$ with the largest degrees as tentative cluster centers at Line 3 when $\mathcal{N}$ is an attributed multiplex graph, we rank the nodes by their summed degrees across all layers.

Lines 12-24 describe the main clustering process of `ANCKA`, which extends the hypergraph-specific Algorithms 1 and 2 with modifications to support attributed graphs and multiplex graphs. First, in orthogonal iterations, calculating $\mathbf{Z}^{(t)}$ is dependent on the type of $\mathcal{N}$. Second, the MHC objective for general networks stems from the analysis in Section 3.4, while the formulation with $\mathbf{P}_N$ is slightly different. In particular, to get MHC ϕ_t without materializing the dense matrix $\mathbf{S}$ in Eq. (3.10) that is expensive to compute, we iteratively obtain ϕ_t via the intermediate matrix $\mathbf{F}^{(\ell)}$ in Eq. (3.27) at Line 21.

$$\mathbf{F}^{(\ell)} = (1-\alpha)((\mathbf{I}-\mathbf{B})\mathbf{P}_N\mathbf{F}^{(\ell-1)} + \mathbf{B}\mathbf{P}_K\mathbf{F}^{(\ell-1)}) + \mathbf{F}^{(0)}, \tag{3.27}$$

where $\mathbf{P}_N$ is Eq. (3.24), (3.25), or (3.26), depending on the type of $\mathcal{N}$. Finally, `ANCKA` adopts the early stopping criteria in Line 24 and returns the clusters with the lowest MHC obtained.

Complexity. When $\mathcal{N}$ is an attributed graph, constructing transition matrix $\mathbf{P}_N$ takes $O(n\bar{\delta})$ time, where $\bar{\delta}$ is the average node degree. For a multiplex network $\mathcal{N}$

Algorithm 4: `ANCKA`

```
Input: Attributed network $\mathcal{N}$ with KNN augmented graph $\mathcal{G}_K$, the number of
       clusters $k$, diagonal matrix $\mathbf{B}$, constant $\alpha$, error threshold $\epsilon_Q$, the
       numbers of iterations $T_a$, $\gamma$, $T_i$, an integer $\tau$.
Output: BCM matrix $\mathbf{Y}$
1  $\mathbf{P}_K \leftarrow \mathbf{D}_K^{-1}\mathbf{A}_K$;
2  Get $\mathbf{P}_N$ by Eq. (3.24), (3.25), or (3.26), depending on the type of $\mathcal{N}$;
3  $\mathcal{V}_c \leftarrow$ sorted indices of $k$ nodes in $\mathcal{N}$ with $k$ largest degrees;
4  Initialize $\mathbf{Z}_0 \leftarrow \mathbf{0}^{k\times n}$;
5  for $j \leftarrow 1$ to $k$ do $\mathbf{Z}_0[j, \mathcal{V}_c[j]] \leftarrow 1$ ;
6  Initialize $\mathbf{\Pi}_c^{(0)} \leftarrow \alpha\mathbf{Z}_0$;
7  for $t \leftarrow 1, 2, \ldots T_i$ do
8      $\mathbf{\Pi}_c^{(t)} \leftarrow (1-\alpha)\mathbf{\Pi}_c^{(t-1)}\mathbf{P}_N + \mathbf{\Pi}_c^{(0)}$;
9  for $v_j \in \mathcal{V}$ do
10     $g(v_j) \leftarrow \arg\max_{1\le l\le k} \mathbf{\Pi}_c^{(T_i)}[l, j]$ ;
11     $\mathbf{Y}^{(0)}[j, g(v_j)] \leftarrow 1$;
12 $\mathbf{Y} \leftarrow \mathbf{Y}^{(0)}$, $\widehat{\mathbf{Y}}^{(0)} \leftarrow h(\mathbf{Y}^{(0)})$;
13 $\mathbf{Q}^{(0)} \leftarrow \frac{1}{\sqrt{n}} \cdot \mathbf{1}|\widehat{\mathbf{Y}}^{(0)}$ ;
14 for $t \leftarrow 1, 2, \cdots, T_a$ do
15     $\mathbf{Z}^{(t)} \leftarrow (\mathbf{I}-\mathbf{B})\mathbf{P}_N \cdot \mathbf{Q}^{(t-1)} + \mathbf{B}\mathbf{P}_K \cdot \mathbf{Q}^{(t-1)}$;
16     $\mathbf{Q}^{(t)}, \mathbf{R}^{(t)} \leftarrow \mathtt{QR}(\mathbf{Z}^{(t)})$ ;
17     if $t$ mod $\tau = 0$ then
18         $\mathbf{Y}^{(t)} \leftarrow \mathtt{Discretize}(\mathbf{Q}^{(t)})$ ;
19         $\widehat{\mathbf{Y}}^{(t)} \leftarrow h(\mathbf{Y}^{(t)})$; $\mathbf{F}^{(0)} \leftarrow \alpha\widehat{\mathbf{Y}}^{(t)}$;
20         for $\ell \leftarrow 1, 2, \ldots \gamma$ do
21             Compute $\mathbf{F}^{(\ell)}$ according to Eq. (3.27)
22         $\Phi(\mathbf{Y}^{(t)}) \leftarrow 1 - \frac{1}{k}trace(\widehat{\mathbf{Y}}^{(t)\top}\mathbf{F}^{(\gamma)})$;
23         if $\Phi(\mathbf{Y}^{(t)}) < \Phi(\mathbf{Y})$ then $\mathbf{Y} \leftarrow \mathbf{Y}^{(t)}$;
24         if Eq. (3.19) or Eq. (3.20) holds then break;
25 return $\mathbf{Y}$;
```

with L layers, the previous results are still valid when L is regarded as constant, as $\mathbf{P}_N$ is aggregated from the transition matrices of all simple graph layers. Given that the number of nonzero entries in $\mathbf{P}_N$ is subject to $O(n\overline{\delta})$, `ANCKA` (Algorithm 4) has the same complexity as Algorithm 1. According to our analysis in Section 3.5.3, the time complexity of `ANCKA` is $O(kn(\overline{\delta}+K+k))$ while its space complexity is $O(n(\overline{\delta}+K+k))$. Since k and K can be viewed as constants, `ANCKA` has space and time complexity of $O(n\overline{\delta})$.

3.7 GPU-Accelerated `ANCKA-GPU`

On large attributed networks, e.g., Amazon and MAG-PM hypergraphs, each with more than 2 million nodes, as reported in Table 3.7, `AHCKA` with 16 CPU threads still needs 1286s and 1372s respectively for clustering, despite its superior efficiency compared with baselines. Moreover, `AHCKA` does not exhibit acceleration proportional to increased CPU threads. As shown in Figure 3.4, when the number of CPU threads is raised from 1 to 32, the time drops from around 3000s to 1200s, with a speedup of merely 2.5 (Amazon) or 2.7 (MAG-PM). In particular, increasing the number of threads from 16 to 32 provides rather limited acceleration (less than 10%).

To overcome the limitation of CPU parallelization, we resort to the massive parallel processing power of GPUs (graphical processing unit) and develop `ANCKA-GPU` to boost efficiency, with about one order of magnitude speedup on large networks with millions of nodes in experiments. For example, `ANCKA-GPU` only needs 120s on an MAG-PM dataset, over 10 times faster than the 1372s of `ANCKA`. Compared to CPUs, the design of GPUs enables them to leverage numerous threads to handle data processing simultaneously, which is beneficial for vector and matrix operations at scale. Please see [19] for details on GPU computing.

As shown in Figure 3.14 of Section 3.8.5 for runtime analysis, the major time-consuming components of ANCKA include invoking Discretize (Line 18 in Algorithm 4), the construction of KNN graph $\mathcal{G}_K$, and expensive matrix operations in orthogonal iterations, greedy initialization and MHC evaluation. With the CuPy library, matrix operations throughout Algorithm 4 can be done on GPUs more efficiently. In the following, we elaborate on the GPU-based discretization and $\mathcal{G}_K$ construction techniques adopted in ANCKA-GPU.

GPU-based Discretization Discretize-GPU. ANCKA uses the off-the-shelf Discretize approach [148] to compute discrete cluster labels $\mathbf{Y}$ from real-valued eigenvectors $\mathbf{Q}$, which could cost substantial time on large datasets. Here, we develop a CUDA kernel Discretize-GPU for efficiency. In what follows, we first explain how the discretization algorithm improves the optimization objective in Definition 2, and then present the design of Discretize-GPU in Algorithm 5.

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

Definition 2. *(Discretization [148]) The solution to the following optimization problem is the optimal discrete* $\mathbf{Y}_{opt}$.

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

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

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

$$\mathbf{Y}[i,g] = \begin{cases} 1, & \text{if } g = \arg\max_{1\le j\le k}(\tilde{\mathbf{Q}}\mathbf{R})[i,j] \\ 0, & \text{otherwise.} \end{cases} \tag{3.28}$$

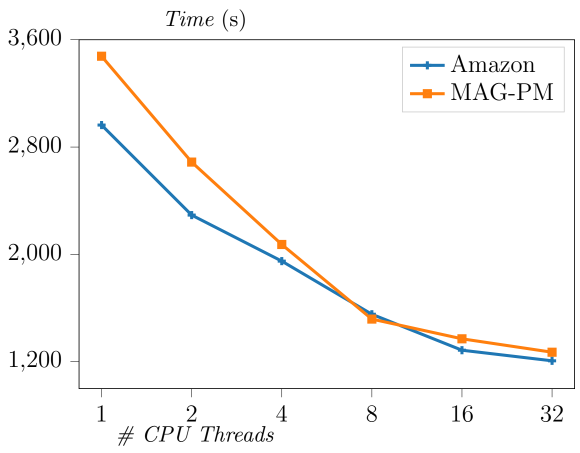


Figure 3.4: Runtime of `AHCKA` with CPU parallelization

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

$$\mathbf{R} = \mathbf{V}\mathbf{U}^\top, \text{ where } \mathbf{U}\mathbf{\Omega}\mathbf{V}^\top \text{ is an SVD of } \tilde{\mathbf{Y}}^\top\tilde{\mathbf{Q}}. \tag{3.29}$$

The iterative process can terminate early when an objective value obj based on $\mathbf{\Omega}$ converges, i.e., its change over the last iteration is within machine precision. This objective is calculated as $obj = n - 2 \times trace(\mathbf{\Omega})$ [148].

We implement the CUDA kernel `Discretize-GPU` in Algorithm 5 to perform the process above to obtain $\mathbf{Y}$. In details, `Discretize-GPU` leverages the grid-block-thread hierarchy of GPU to assign threads to handle $n \times k$ matrices, including $\mathbf{Q}$ and $\mathbf{Y}$. Each row in such a matrix is processed by a block of threads, identified by a block id bid; each of the k elements in the row is handled by a thread tid in the block. Consequently, given a matrix $\mathbf{Q}$, we can use $\mathbf{Q}[bid, tid]$ to represent that the corresponding element in $\mathbf{Q}$ is handled by the tid-th thread in block bid on a GPU. Parallel row normalization is performed at Lines 1-2 to get $\tilde{\mathbf{Q}}$. After initializing $\mathbf{R}$ as a $k \times k$ identity matrix (Line 3), we alternately update $\tilde{\mathbf{Y}}$ and $\mathbf{R}$ for at most max_iter iterations (Lines 4-12) and terminate early when the objective value obj does not change over the current iteration at Line 12. Within an iteration, we first

Algorithm 5: Discretize-GPU

```
Input: eigenvector matrix $\mathbf{Q}$
Output: Intermediate BCM matrix $\mathbf{Y}$
1  Parallel for $i \leftarrow 1, 2, \cdots, n$ do
2      $\tilde{\mathbf{Q}}[i] \leftarrow \frac{\mathbf{Q}[i]}{\|\mathbf{Q}[i]\|_2}$ ;
3  $\mathbf{R} \leftarrow \mathbf{I}_k$ ;
4  while $iter \leftarrow 1, 2, \cdots, max\_iter$ do
5      Update $\mathbf{Y}$ by Eq. (3.28) via argmax kernel on GPU;
6      Parallel for $j \leftarrow 1, 2, \cdots, k$ do
7          $col\_sum[j] \leftarrow \sum_{i=1}^{n} \mathbf{Y}[i, j]$
8      Parallel for each $tid < k$ in blocks do
9          $\tilde{\mathbf{Y}}[bid, tid] \leftarrow \frac{\mathbf{Y}[bid,tid]}{col\_sum[tid]}$;
10     $\mathbf{U}, \mathbf{\Omega}, \mathbf{V}^{\top} \leftarrow$ SVD_GPU($\tilde{\mathbf{Y}}^{\top}\tilde{\mathbf{Q}}$) ;
11     $\mathbf{R} \leftarrow \mathbf{V}\mathbf{U}^{\top}$ on GPU;
12     if Objective value obj does not change then break;
13 return $\mathbf{Y}$;
```

update $\mathbf{Y}$ at Line 5, then perform column normalization to get $\tilde{\mathbf{Y}}$ (Lines 6-9), and then perform SVD on GPU over $\tilde{\mathbf{Y}}^{\top}\tilde{\mathbf{Q}}$ to get $\mathbf{U}$ and $\mathbf{V}$ at Line 10, which helps to update $\mathbf{R}$ at Line 11. Finally, $\mathbf{Y}$ is returned at Line 13.

KNN construction. An $n \times d$ attributed matrix $\mathbf{X}$ requires KNN search on its rows to construct the augmented graph $\mathcal{G}_K$ and thus the transition matrix $\mathbf{P}_K$. For this purpose, we adopt Faiss [50], a GPU-compatible similarity search library. In Algorithm 6 for $\mathcal{G}_K$ construction, we first normalize all rows in $\mathbf{X}$ at Lines 1-2 to facilitate the computation of cosine similarity between row vectors. Faiss supports various indexes for KNN computation, and the index type suitable for ANCKA is determined based on the input data volume. For small or medium datasets where the number of nodes $|\mathcal{V}|$ is below 100,000, since the time cost for exact similarity search is affordable, we choose the flat index with a plain encoding of each row vector in $\mathbf{X}$,

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

```
Input: Network $\mathcal{N}$, attribute matrix $\mathbf{X}$, parameter $K$.
Output: KNN transition matrix $\mathbf{P}_K$
1  Parallel for $i \leftarrow 1, 2, \cdots, n$ do
2      $\mathbf{X}[i] \leftarrow \frac{\mathbf{X}[i]}{||\mathbf{X}[i]||_2}$;
3  if $|\mathcal{V}| < 100,000$ then
4      $index \leftarrow$ FlatIndex ($\mathbf{X}$) ;
5  else
6      $index \leftarrow$ IVFPQIndex ($\mathbf{X}$) ;
7  Invoke Faiss on GPU to get the KNN of each row in $\mathbf{X}$ ;
8  Get $\mathbf{A}_K$ by Eq. (3.1) on GPU ;
9  $\mathbf{D}_K \leftarrow$ Diag($\mathbf{A}_K \mathbf{1}_n$) on GPU ;
10 $\mathbf{P}_K \leftarrow \mathbf{D}_K^{-1} \mathbf{A}_K$ on GPU;
11 return $\mathbf{P}_K$;
```

to achieve exact KNN computation (Lines 3-4). Otherwise, we turn to approximate nearest neighbor search on large datasets with the IVFPQ index that combines the inverted file index (IVF) with the product quantization (PQ) technique at Line 6. In particular, IVF index narrows down the search to closely relevant partitions that contain the nearest neighbors at a high probability, while PQ produces memory-efficient encoding of attribute vectors. Faiss on GPU is invoked to get the KNN of each row in $\mathbf{X}$, and $\mathbf{A}_K$ is obtained by Eq. (3.1) at Lines 7-8. Then, the degree matrix $\mathbf{D}_K$ and transition matrix $\mathbf{P}_K$ are computed on GPU (Lines 9-10) and returned at Line 11.

3.8 Experiments

We experimentally evaluate the proposed `ANCKA` and competitors in terms of both clustering quality and efficiency. We also evaluate the performance of `ANCKA-GPU`

Table 3.1: Dataset statistics.

Task	Dataset	Type	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	d	k
AHC	Query	HG	481	15,762	426	6
	Cora-CA	HG	2,708	1,072	1,433	7
	Cora-CC	HG	2,708	1,579	1,433	7
	Citeseer	HG	3,312	1,079	3,703	6
	20News	HG	16,242	100	100	4
	DBLP	HG	41,302	22,363	1,425	6
	Amazon	HG	2,268,083	4,285,295	1,000	15
	MAG-PM	HG	2,353,996	1,082,711	1,000	22
AGC	Cora	UG	2,708	5,429	1,433	7
	Citeseer-UG	UG	3,327	4,732	3,703	6
	Wiki	UG	2,405	17,981	4,973	17
	Citeseer-DG	DG	3,312	4,715	3,703	6
	TWeibo	DG	2,320,895	50,655,143	1,657	8
	Amazon2M	UG	2,449,029	61,859,140	100	47
AMGC	ACM	MG	3,025	29,281 2,210,761	1,870	3
	IMDB	MG	3,550	13,788 66,428	2,000	3
	DBLP-MG	MG	4,057	11,113 5,000,495 7,043,571	334	4

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

3.8.1 Experimental Setup

Datasets

Table 3.1 provides the statistics of 17 real-world attributed networks used in experiments, including attributed hypergraphs (HG), undirected graphs (UG), directed graphs (DG), and multiplex graphs (MG). $|\mathcal{V}|$ and $|\mathcal{E}|$ are the number of nodes and edges (or hyperedges), respectively, d is the attribute dimension and k is the number of ground-truth clusters.

We gather 8 attributed hypergraph datasets. Query dataset [123] is a Web query hypergraph, where nodes represent queries and are connected by hyperedges representing query sessions, and nodes are associated with attributes of keyword embeddings and associated webpages. Cora-CA, Cora-CC, Citeseer, and DBLP are four benchmark datasets used in prior work [136]. All of them are originally collected from academic databases, where each node represents a publication, node attributes are binary word vectors of abstract, and research topics are regarded as ground-truth clusters. Hyperedges correspond to co-authorship in Cora-CA and DBLP datasets or co-citation relationship in Cora-CC and Citeseer datasets. 20News dataset [43] consists of messages taken from Usenet newsgroups. Messages are nodes, and the messages containing the same keyword are connected by a corresponding hyperedge, and the TF-IDF vector for each message is used as the node attribute. Amazon dataset is constructed based on the 5-core subset of Amazon reviews dataset [90], where each node represents a product and a hyperedge contains the products reviewed by a user. For each product, we use the associated textual metadata as the node attributes and the product category as its cluster label. MAG-PM dataset is extracted from the Microsoft Academic Graph [110], where nodes, co-authorship hyperedges, attributes, and cluster labels are obtained as in other academic datasets (i.e., Cora-CA, Cora-CC, Citeseer, and DBLP).

In Table 3.1, we also consider 6 attributed graphs, which are commonly used for AGC [154, 52, 144, 15]. Cora, Citeseer-UG, Wiki, and Amazon2M are undirected, while Citeseer-DG and TWeibo are directed. TWeibo [144] and Amazon2M [15] are two large-scale attributed graphs. TWeibo is a social network where each node represents a user, and the directed edges represent relationships between users. Amazon2M is constructed based on the co-purchasing networks of products on Amazon. Cora, Citeseer-UG, and Citeseer-DG are citation networks where nodes represent publications, a pair of nodes are connected if one cites the other, and nodes are associated with binary word vectors as features. Wiki is a webpage network where each edge in the graph indicates that one webpage is linked to the other, while the node attributes are TF-IDF feature vectors. Moreover, three attributed multiplex graphs, namely ACM, IMDB, and DBLP-MG, are considered for AMGC [49, 91, 77]. ACM is an academic publication network comprising co-author and co-subject graph layers, as well as bag-of-words attributes of keywords. IMDB is a movie network with plot text embeddings as attributes and two graph layers representing the co-director (directed by the same director) and co-actor (starring the same actor) relations, respectively. DBLP-MG is a researcher network including publication keyword vectors as attributes and three graph layers: co-author, co-conference (publishing at the same conference), and co-term (sharing common key terms). ACM and DBLP-MG have research areas labeled as ground truth clusters, while IMDB is labeled by movie genres.

Competitors and Parameter Settings

The 19 competitors for AHC are summarized as follows:

- 3 plain hypergraph clustering methods including `HNCut` [156], `HyperAdj` [102], and `KaHyPar` [35];

- the extended AHC versions of the 3 methods above (dubbed as `ATHNCut`, `ATHyperAdj`, and `ATKaHyPar`), which work on an augmented hypergraph with attribute-KNN hyperedges of all nodes merged into the input hypergraph;

- `ATMetis` that applies the traditional graph clustering algorithm `Metis` [55] over a graph constructed by clique expansion of the input hypergraph and attribute KNN graph augmentation; `Infomap` [103], `Louvain` [8], `k-MQI` and `k-Nibble` (extended from `MQI` [63] and PageRank-Nibble [4] for k-way clustering via $k-1$ consecutive bisections as described in technical report [71]) on the same KNN-augmented clique-expansion graph;

- 3 AHC algorithms including the recent `GRAC` [27] and NMF-based approaches (`GNMF` [10, 27] and `JNMF` [23]);
- `ACMin-C` and `ACMin-S`, obtained by applying an attributed graph clustering method `ACMin` [144] over the graphs reduced from hypergraphs by clique expansion and star expansion, respectively; probabilistic model `CESNA` [139] with clique-expansion;
- `k-means` and `HAC` (hierarchical agglomerative clustering [121]) algorithms applied to the node attribute matrix.

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

- 6 AGC approaches including NMF-based algorithm `GNMF` [10], graph convolution algorithm `AGCGCN` [154] , probabilistic model `CESNA` [139], spectral clustering on fine-grained graphs method `FGC` [53], attributed random walk approach `ACMin` [144], and the clustering framework `GRACE` [52] generalized from `GRAC`.
- `NCut` [106] and `Metis` [55] that are conventional graph clustering methods applied to the input graph;
- `ATNCut` and `ATMetis` that are `NCut` and `Metis` applied to the augmented graph with attribute KNN; `Infomap` [103], `Louvain` [8], `k-MQI` [63] and `k-Nibble` [4] on the augmented graph with attribute KNN.

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

- 5 AMGC methods: a multi-view graph auto-encoder model `O2MAC` [24], `HDMI` [49] that learns node embeddings via higher-order mutual information loss, `MCGC` [91] and `MAGC` [77] which perform graph filtering and find a consensus graph for spectral clustering, and `GRACE` [52] that is a general graph convolution clustering method;

Table 3.2: Attributed Hypergraph Clustering (AHC) Quality on Small Datasets.

	Query				Cora-CA				Cora-CC				Citeseer			
Algorithm	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI
`HyperAdj`	0.212	0.198	0.013	-0.004	0.233	0.216	0.038	0.022	0.255	0.191	0.039	0.015	0.226	0.182	0.008	0.002
`HNCut`	0.239	0.218	0.016	0.002	0.238	0.127	0.023	-0.002	0.213	0.125	0.021	-0.005	0.222	0.167	0.010	0.004
`KaHyPar`	0.220	0.205	0.016	0.003	0.275	0.265	0.084	0.050	0.309	0.289	0.135	0.089	0.275	0.265	0.045	0.036
`k-means`	0.586	0.581	0.461	0.230	0.349	0.297	0.158	0.086	0.351	0.312	0.176	0.097	0.460	0.424	0.219	0.185
`HAC`	0.541	0.575	0.453	0.173	0.374	0.336	0.234	0.096	0.374	0.336	0.234	0.096	0.376	0.352	0.188	0.083
`ATHyperAdj`	0.281	0.259	0.036	0.019	0.255	0.232	0.061	0.032	0.262	0.238	0.061	0.035	0.218	0.198	0.010	0.005
`ATHNCut`	0.241	0.220	0.017	0.003	0.438	0.297	0.263	0.183	0.556	0.456	0.317	0.288	0.563	0.483	0.325	0.286
`ATMetis`	0.520	0.507	0.349	0.264	0.575	0.550	0.403	0.346	0.552	0.529	0.379	0.310	0.612	0.590	0.357	0.348
`ATKaHyPar`	0.243	0.225	0.025	0.009	0.528	0.477	0.316	0.260	0.529	0.480	0.299	0.246	0.551	0.532	0.304	0.276
`k-MQI`	0.222	0.071	0.019	-0.001	0.304	0.070	0.005	0.001	0.302	0.069	0.005	0.000	0.212	0.059	0.003	0.000
`k-Nibble`	0.252	0.121	0.025	0.008	0.321	0.119	0.070	0.060	0.391	0.165	0.155	0.098	0.345	0.170	0.139	0.102
`Infomap`	0.235	0.215	0.017	0.002	0.514	0.464	0.343	0.266	0.541	0.479	0.393	**0.347**	0.491	0.463	0.263	0.221
`Louvain`	0.239	0.218	0.017	0.003	0.501	0.430	0.332	0.217	0.569	**0.546**	0.373	0.269	0.570	0.486	0.319	0.308
`CESNA`	0.222	0.191	0.024	0.002	0.305	0.092	0.030	0.000	0.378	0.240	0.140	0.053	0.206	0.060	0.012	0.000
`ACMin-C`	0.233	0.219	0.017	0.003	0.526	0.493	0.319	0.237	0.556	0.473	0.349	0.259	0.643	0.587	0.355	0.376
`ACMin-S`	0.241	0.140	0.008	-0.002	0.523	0.477	0.318	0.239	0.526	0.462	0.340	0.249	0.636	0.597	0.351	0.365
`GNMF`	0.451	0.413	0.345	0.247	0.460	0.412	0.240	0.165	0.436	0.355	0.194	0.132	0.500	0.462	0.271	0.257
`JNMF`	0.216	0.211	0.014	-0.001	0.494	0.443	0.286	0.216	0.453	0.426	0.230	0.178	0.543	0.518	0.246	0.242
`GRAC`	0.410	0.389	0.196	0.087	0.601	0.593	0.376	0.308	0.556	0.507	0.349	0.262	0.612	0.575	0.329	0.332
`ANCKA`	**0.715**	**0.662**	**0.645**	**0.571**	**0.651**	**0.608**	**0.462**	**0.406**	**0.592**	0.520	**0.412**	0.338	**0.662**	**0.615**	**0.392**	**0.397**

- `NCut` [106] and `Metis` [55] that apply traditional graph clustering methods over the aggregation of the adjacency matrices of all graph layers in the input multiplex graph;

- `ATNCut` and `ATMetis` that apply `NCut` and `Metis` to the aggregated matrix of all layers' adjacency matrices and the attribute KNN graph; `Infomap` [103], `Louvain` [8], `k-MQI` [63] and `k-Nibble` [4] in the same way;

- `CESNA` [139] that treats the aggregated adjacency matrix of all layers as an attributed graph;

For all competitors, we adopt the default parameter settings as suggested in their respective papers. Hyperparameters for AMGC algorithms `MCGC` and `MAGC` are tuned as instructed in the corresponding papers, and we report the best results acquired. As for `ANCKA` on attributed hypergraphs, i.e., `AHCKA` [73], unless otherwise specified, we set parameters on all datasets: $\alpha = 0.2$, $\beta = 0.5$, and $\gamma = 3$, parameter $K = 10$ for KNN construction, the convergence threshold $\epsilon_Q = 0.005$, and the numbers of iterations $T_a = 1000$, $T_i = 25$. The interval parameter τ is set to 5 on all datasets

Table 3.3: Attributed Hypergraph Clustering (AHC) Quality on Medium/Large Datasets.

	20News				DBLP				Amazon				MAG-PM				**Quality**
Algorithm	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	**Rank**
HyperAdj	0.338	0.274	0.010	0.010	0.234	0.158	0.019	0.007	0.292	0.105	0.043	0.070	0.138	0.078	0.051	0.028	15.5
HNCut	0.683	0.561	0.373	0.387	0.279	0.113	0.020	0.009	0.310	0.032	0.001	0.000	0.253	0.022	0.005	0.001	13.8
KaHyPar	0.479	0.468	0.169	0.172	0.559	0.534	0.390	0.338	0.494	0.442	**0.694**	0.385	0.367	0.306	0.483	0.247	10.9
k-means	0.404	0.373	0.147	0.045	0.529	0.513	0.362	0.283	0.380	0.257	0.362	0.175	0.272	0.196	0.229	0.071	10.1
HAC	0.430	0.382	0.237	0.058	0.571	0.532	0.372	0.310	OOM				OOM				11.1
ATHyperAdj	0.317	0.261	0.016	0.006	0.296	0.220	0.068	0.035	0.273	0.103	0.050	0.057	0.189	0.048	0.043	-0.006	13.8
ATHNCut	0.338	0.133	0.002	0.001	0.458	0.245	0.386	0.173	0.310	0.033	0.003	0.000	0.269	0.035	0.035	-0.001	10.8
ATMetis	0.612	0.596	0.264	0.281	0.671	0.670	0.567	0.496	OOM				0.304	0.254	0.401	0.196	4.9
ATKaHyPar	0.632	0.610	0.295	0.328	0.650	0.658	0.522	0.457	0.527	**0.504**	0.680	0.386	0.352	0.295	0.411	0.205	5.7
k-MQI	0.336	0.126	0.000	0.000	0.271	0.071	0.000	0.000	OOM				0.252	0.018	0.000	0.000	17.1
k-Nibble	0.338	0.129	0.002	0.000	0.254	0.086	0.028	0.006	OOM				0.252	0.019	0.000	0.000	14.3
Infomap	0.338	0.129	0.004	0.000	0.595	0.573	0.488	0.404	OOM				0.398	0.172	0.380	0.248	9.5
Louvain	0.633	0.522	0.304	0.323	0.643	0.580	0.554	0.470	OOM				OOM				8.3
CESNA	0.379	0.350	0.086	0.047	0.272	0.072	0.001	0.000	>12h				>12h				15.5
ACMin-C	0.558	0.524	0.219	0.239	0.607	0.563	0.503	0.445	0.458	0.113	0.354	0.244	0.519	0.293	0.499	0.430	6.3
ACMin-S	0.7116	**0.669**	0.365	0.416	0.547	0.474	0.472	0.359	0.473	0.056	0.393	0.263	0.550	0.341	0.550	**0.499**	6.8
GNMF	0.436	0.271	0.070	0.061	0.613	0.506	0.417	0.407	OOM				OOM				10.6
JNMF	0.582	0.423	0.247	0.241	0.618	0.588	0.447	0.396	OOM				OOM				11.0
GRAC	0.391	0.306	0.068	0.056	0.648	0.657	0.563	0.487	0.612	0.488	0.625	0.486	0.398	0.315	0.386	0.197	5.3
ANCKA	**0.7118**	0.658	**0.409**	**0.469**	**0.797**	**0.774**	**0.632**	**0.632**	**0.660**	0.492	0.630	**0.524**	**0.566**	**0.405**	**0.561**	0.471	**1.3**

except the large and dense hypergraph Amazon, where we set $\tau = 1$ to expedite early termination in light of the immense per-iteration overhead when processing Amazon. On large datasets (i.e., Amazon and MAG-PM), T_i is set to 1 and $\beta = 0.4$. In ANCKA, for attributed graphs and multiplex graphs, we fix $K = 50$, except for large datasets TWeibo and Amazon2M with $K = 10$. In particular, we find it necessary to adjust the β parameter for certain instances following the practice in recent works [52, 91, 77]. β is set to 0.5 for Cora and Wiki and 0.4 on Citeseer-UG, Citeseer-DG, TWeibo, and Amazon2M. We tune β in $[0.1, 0.9]$ by step size 0.1 for multiplex graphs.

All the remaining hyperparameters in ANCKA follow the default setting of AHCKA. The parameter settings in GPU-based ANCKA-GPU are identical to ANCKA.

3.8.2 Performance Evaluation

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

Table 3.4: Attributed Graph Clustering (AGC) Quality on Cora, Citeseer-UG & Wiki.

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

Table 3.5: Attributed Graph Clustering (AGC) Quality on Citeseer-DG, Tweibo & Amazon2M.

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

Quality Evaluation

The clustering quality is measured by 4 classic metrics including overall accuracy (Acc), average per-class F1 score (F1), normalized mutual information (NMI), and adjusted Rand index (ARI). The former three metrics are in the range $[0, 1]$, whereas

Table 3.6: Attributed Multiplex Graph Clustering (AMGC) Quality.

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

Table 3.7: Efficiency of Attributed Hypergraph Clustering (AHC) (Time in Seconds, RAM in GBs). The Quality Rank column is from Table 3.3. Among all native AHC methods in the last 4 rows, the best is in bold, and the runner-up is underlined.

	Query		Cora-CA		Cora-CC		Citeseer		20News		DBLP		Amazon		MAG-PM		Quality
Algorithm	Time	RAM	Time	RAM	Time	RAM	Time	RAM	Time	RAM	Time	RAM	Time	RAM	Time	RAM	Rank
`HNCut`	0.057	0.139	0.582	0.141	0.654	0.158	0.528	0.172	0.116	0.134	4.282	0.625	477.1	3.291	666.8	4.256	13.8
`KaHyPar`	1.556	0.105	0.375	0.132	0.285	0.129	0.292	0.159	1.516	0.119	3.795	0.606	3707	37.14	556.9	9.487	10.9
`ATHNCut`	0.393	0.178	0.650	0.428	0.657	0.429	0.835	1.031	4.935	2.164	35.68	3.589	685.9	54.24	789.3	57.36	10.8
`ATMetis`	0.081	0.125	0.238	0.282	0.239	0.283	0.389	0.307	13.55	2.966	26.85	3.308	OOM		557.9	64.20	4.9
`ATKaHyPar`	1.668	0.128	1.610	0.225	1.543	0.225	1.733	0.304	11.47	2.400	50.32	3.256	5529	54.28	1509	57.41	5.7
`k-MQI`	0.104	0.243	0.361	0.352	0.376	0.363	0.418	0.427	11.23	3.866	28.98	3.397	OOM		1567	60.54	17.1
`k-Nibble`	0.151	0.224	5.827	0.655	5.888	0.667	21.83	0.885	44.55	8.975	1338	51.77	OOM		3858	281.6	14.3
`Infomap`	0.221	0.191	0.742	0.291	0.611	0.293	0.719	0.363	556.1	21.43	43.50	3.263	OOM		11756	200.9	9.5
`Louvain`	0.732	0.195	1.915	0.232	0.735	0.242	1.911	0.313	1567	21.77	70.15	3.313	OOM		OOM		8.3
`CESNA`	0.620	0.119	2.400	0.134	9.816	0.137	3.251	0.164	7643	0.157	92.04	0.617	>12h		>12h		15.5
`GNMF`	2.851	0.494	15.92	0.369	20.55	0.316	72.96	0.562	36.76	4.273	612.3	33.27	OOM		OOM		10.6
`JNMF`	3.366	0.494	7.754	0.369	22.66	0.317	61.32	0.549	253.3	4.273	3247	33.27	OOM		OOM		11.0
`GRAC`	1.701	**0.142**	7.661	0.288	3.696	0.287	13.15	0.454	**3.368**	**0.275**	91.21	1.700	14662	175.1	3504	92.69	5.3
`ANCKA`	**0.342**	0.161	**0.402**	**0.231**	**0.416**	**0.232**	**0.635**	**0.317**	8.176	0.383	**41.50**	**0.998**	**1286**	**56.71**	**1371**	**59.25**	**1.3**

ARI ranges from -0.5 to 1. We also sort all methods by each metric and calculate their average Quality Rank for AHC, AGC, and AMGC, provided in the last column of Tables 3.3, 3.5 and 3.6.

AHC. Tables 3.2 and 3.3 present the Acc, F1, NMI, and ARI scores of each method on small and medium/large attributed hypergraph datasets, respectively. The first observation from Tables 3.2 and 3.3 is that `ANCKA` on attributed hyergraphs (i.e., `AHCKA`) consistently achieves outstanding performance over all competitors on all datasets under almost all metrics, often by a significant margin. ANCKA has a quality rank of 1.3, much higher than the runner-up ATMetis (4.9) and GRAC (5.2). On all the four small datasets (*i.e.*, Query, Cora-CA, Cora-CC, and Citeseer), ANCKA outperforms the best competitors (underlined in Table 3.2) by at least 1.9% in terms of Acc and NMI. On all the four medium/large attributed hypergraphs (*i.e.*, 20News, DBLP, Amazon, and MAG-PM), ANCKA also yields remarkable improvements upon the competitors, with percentages up to 12.6%, 10.4%, 6.5%, 13.6% in Acc, F1, NMI, and ARI respectively. Few exceptions exist, where ANCKA still leads in three out of the four metrics, demonstrating the best overall performance. The results in Tables 3.2 and 3.3 also confirm the effectiveness of ANCKA over various attributed hypergraphs from different application domains, *e.g.*, web queries, news messages, and review data. The performance of ANCKA is ascribed to our optimizations based on KNN augmentation and MHC in Section 3.3 and Section 3.4, and the framework for generating high-quality BCM matrices in Section 3.5.

AGC. Tables 3.4 and 3.5 present the Acc, F1, NMI, and ARI scores of each method on all attributed graphs for AGC task. ANCKA consistently outperforms existing competitors under most metrics, though few exceptions exist where ANCKA is comparable to the best. ANCKA has a quality rank of 1.3, much higher than the runner-up with quality rank 4.0. For example, on Citeseer-UG in Table 3.4, ANCKA achieves higher Acc, F1, NMI and ARI than the runner-up performance underlined. On the two large datasets, TWeibo and Amazon2M in Table 3.5, ANCKA also produces clusters with high quality, while GNMF, AGCGCN, and FGC run out of memory or cannot finish within 12 hours. Notably, on Amazon2M, ANCKA surpasses all methods on all metrics except F1 (0.006 behind ATMetis) while achieving 0.494 accuracy (runner-up is Louvain at 0.463) and 0.545 ARI (runner-up is Louvain at 0.520). The effectiveness of ANCKA validates the versatility of the proposed techniques for different clustering tasks, e.g., AGC. Besides, ATMetis and ATNCut generally outperform Metis and NCut in AGC performance, respectively, exhibiting the efficacy of the proposed KNN augmentation.

Table 3.8: Efficiency of Attributed Graph Clustering (AGC) Algorithms (Time in Seconds, RAM in GBs).The Quality Rank column is from Table 3.5. Among all native AGC methods in the last 7 rows, the best is in bold, and the runner-up is underlined.

	Cora		Citeseer-UG		Wiki		Citeseer-DG		Tweibo		Amazon2M		Quality
Algorithm	Time	RAM	Time	RAM	Time	RAM	Time	RAM	Time	RAM	Time	RAM	Rank
`Metis`	0.006	0.203	0.009	0.347	0.242	0.489	0.006	0.316	121.6	4.688	46.40	8.150	10.0
`NCut`	0.072	0.198	0.350	0.326	0.087	0.475	0.321	0.347	409.3	4.662	874.8	8.132	13.8
`ATMetis`	0.688	0.352	0.571	0.533	0.807	0.591	0.398	0.275	360.0	13.58	130.4	16.63	6.8
`ATNCut`	0.469	0.351	0.589	0.501	0.902	0.560	0.548	0.258	334.6	13.66	502.5	16.58	10.2
`k-MQI`	0.382	0.438	0.488	0.585	0.721	0.502	0.348	0.447	2442	29.90	1453	19.65	14.3
`k-Nibble`	4.656	0.495	19.21	0.626	13.86	0.550	18.99	0.685	3826	102.3	3587	89.14	12.0
`Infomap`	1.265	0.398	1.668	0.544	1.499	0.739	1.556	0.310	6701	97.48	4155	45.75	6.6
`Louvain`	6.773	0.371	7.319	0.503	4.527	0.669	6.584	0.572	10010	84.50	21696	72.80	4.0
`CESNA`	12.81	**0.144**	35.21	**0.167**	354.9	**0.162**	28.12	**0.178**	>12h		<u>1931</u>	**7.701**	12.8
`GNMF`	13.18	0.269	37.01	0.397	22.38	0.579	42.81	0.438	OOM		OOM		9.6
`AGCGCN`	5.842	0.960	33.34	2.120	5.965	1.003	34.18	2.326	OOM		OOM		8.3
`FGC`	29.68	1.998	225.7	3.273	50.93	3.080	44.93	3.571	>12h		>12h		6.5
`ACMin`	**0.368**	<u>0.164</u>	**0.400**	<u>0.177</u>	<u>3.646</u>	<u>0.380</u>	**0.556**	<u>0.234</u>	**1098**	**18.61**	5300	20.21	5.6
`GRACE`	5.589	0.651	21.82	1.793	16.78	1.740	15.23	1.960	2317	60.44	3162	39.71	<u>4.0</u>
`ANCKA`	<u>1.251</u>	0.369	<u>1.587</u>	0.517	**0.907**	0.706	<u>0.838</u>	0.280	<u>1318</u>	<u>19.89</u>	**1708**	<u>17.01</u>	**1.3**

AMGC. Table 3.6 reports the Acc, F1, NMI, and ARI scores of all methods on all attributed multiplex graphs. `ANCKA` has the best quality rank. As shown, on ACM and DBLP-MG, `ANCKA` achieves the best clustering quality among all methods under all metrics, with NMI and ARI leading by at least 3% on ACM, while being the second best in three metrics on IMDB. As shown later in Table 3.9, on these datasets, `ANCKA` is faster than existing native AMGC methods by at least an order of magnitude. With the intuitive design of random walk transition matrix $\mathbf{P}_N$ on multiplex graphs in Section 3.6.1, `ANCKA` can utilize the proposed KNN augmentation, clustering objective, and optimization techniques to maintain its excellent performance on the AMGC task.

Efficiency Evaluation

Tables 3.7, 3.8 and 3.9 report the runtime (in seconds, with KNN construction included) and memory overhead (in Gigabytes), for AHC, AGC, and AMGC, respectively. For ease of comparing the trade-off between quality and efficiency, the last column of Tables 3.7, 3.8 and 3.9 contains the corresponding quality ranks from Tables 3.3, 3.5 and 3.6, respectively. In each table, the methods are separated into two

Table 3.9: Efficiency of Attributed Multiplex Graph Clustering (AMGC) Algorithms (Time in Seconds, RAM in GBs). The Quality Rank column is from Table 3.6. Among all native AMGC methods in the last 6 rows, the best is in bold, and the runner-up is underlined.

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

categories: *non-native* methods extended from other clustering problems and *native* methods for the corresponding task. For instance, in Table 3.7, there are 4 native AHC methods in the last 4 rows, while the non-native methods are in the rows above.

In Tables 3.7, 3.8, and 3.9, although certain non-native methods are efficient, their quality ranks in terms of clustering quality are typically low. Hence, in the following, we mainly compare the efficiency of `ANCKA` against the native methods for each task. A method is terminated early if it runs out of memory (OOM) or cannot finish within 12 hours.

AHC. In Table 3.7, compared with native AHC methods, we can observe that `ANCKA` is significantly faster on most datasets, often by orders of magnitude. For example, on a small graph Citeseer, `ANCKA` takes 0.635 seconds, while the fastest AHC competitor `GRAC` needs 13.15 seconds, meaning that `ANCKA` is 20.7× faster. On large attributed hypergraphs including Amazon and MAG-PM, most existing AHC solutions fail to finish due to the OOM errors, whereas `ANCKA` achieves 11.4× and 2.6× speedup over the only viable native AHC competitor GRAC on Amazon and MAG-PM, respectively. An exception is 20News, which contains a paucity of hyperedges (100 hyperedges), where ANCKA is slower than GRAC. Recall that in Table 3.3, compared to ANCKA, GRAC yields far inferior accuracy in terms of clustering on 20News, which highlights the advantages of ANCKA over GRAC. Additionally, while ATMetis is fast, it achieves an average quality rank of 4.9, which falls short of the 1.7 quality rank attained by ANCKA. As shown in Tables 3.2 and 3.3, ANCKA surpasses ATMetis in all metrics but one. Moreover, ATMetis encounters OOM on Amazon. As for the memory consumption (including the space to store hypergraphs), observe that ANCKA has comparable memory overheads with the native AHC competitors on small graphs and up to 3.1× memory reduction on medium/large graphs.

AGC. In Table 3.8 for AGC, ANCKA has comparable running time to ACMin, a recent AGC method that is optimized for efficiency, while being faster than the other native AGC methods. However, the quality rank of ANCKA is 1.3, much higher than 5.6 of ACMin. Specifically, in Tables 3.4 and 3.5, ANCKA consistently achieves better clustering quality than ACMin on all six attributed graphs under all metrics. Moreover, ANCKA remains to be the runner-up in terms of running time on the first five datasets, and is the fastest on the largest Amazon2M for clustering. Memory-wise, ANCKA consumes a moderate amount of memory that stays below 1GB over the first four small datasets and achieves decent performance on two large datasets, TWeibo and Amazon2M.

AMGC. In Table 3.9, ANCKA achieves a significant speedup ratio over the native AMGC baselines, often by an order of magnitude, while being memory efficient. Specifically, ANCKA achieves a speedup of 15.0×, 13.9×, and 9.5×, compared to the runner-up native AMGC methods MAGC and GRACE. The memory consumption of ANCKA is also less than the majority of existing native AMGC methods.

Table 3.10: Evaluation between `ANCKA` and `ANCKA-GPU`.

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

Evaluation on `ANCKA-GPU`

We compare the cluster quality and efficiency of the CPU-based `ANCKA` against `ANCKA-GPU` in Section 3.7, with results reported in Table 3.10 for the three tasks (AHC, AGC, and AMGC) over all datasets. First, observe that `ANCKA-GPU` achieves similarly high-quality cluster results as the CPU-based `ANCKA` across all datasets for all three tasks, and the quality difference between `ANCKA-GPU` and `ANCKA` are often negligible, in terms of Acc, F1, NMI, and ARI.

The last column of Table 3.10 provides the running time of `ANCKA-GPU` and `ANCKA` with 16 CPU threads. For the AHC task, the speedup of `ANCKA-GPU` is less significant on the small attributed hypergraphs (Query, Cora-CA, Cora-CC, and Citeseer). We ascribe this to the numerous SVD operations on small $k \times k$ matrices in `Discretize-GPU`, as it has been known that small dimensions of input matrices may hurt the efficiency of GPU-based SVD [3]. On medium/large attributed hypergraphs (20News, DBLP, Amazon, and MAG-PM), the GPU-accelerated version, `ANCKA-GPU`, achieves speedup ratios of 30.5, 70.2, 8.44, and 11.4, respectively, over the CPU version `ANCKA`. The high speedup ratios of `ANCKA-GPU`, often exceeding an order of magnitude, validate the efficiency of the technical designs elaborated in Section 3.7, especially on large-scale hypergraphs. For the AGC task, similarly, on small attributed graphs, Cora, Citeseer-UG, Wiki, and Citeseer-DG, `ANCKA-GPU` is faster than `ANCKA` while the speedup ratio is usually below 10, due to the same reason explained above. On large attributed graphs (TWeibo and Amazon2M), `ANCKA-GPU` is more efficient than `ANCKA` by an order of magnitude. For the AMGC task, `ANCKA-GPU` is also consistently faster than `ANCKA` on all attributed multiplex graphs. The memory consumption of `ANCKA-GPU` is measured by GPU video memory (VRAM), while that of `ANCKA` is by RAM, and the consumption is reported in the second last column of Table 3.10 in GBs. The memory usage of `ANCKA-GPU` and `ANCKA` is not directly comparable, due to the different computational architectures and libraries used on GPUs and CPUs. Note that the major memory consumption of our implementations is in the KNN augmentation step. On small or medium-sized datasets, e.g., Query and Cora-CA, VRAM usage by `ANCKA-GPU` is higher than the RAM usage by `ANCKA`. The reason is that `ANCKA-GPU` uses GPU-based Faiss for nearest-neighbor search and Faiss allocates about 700MB of VRAM for temporary storage. On large datasets, `ANCKA` requires a substantial RAM space due to the implementation of the ScaNN algorithm for KNN, while GPU-based Faiss in `ANCKA-GPU` requires less VRAM space.

Then we enhance `GRACE` [52] with GPU acceleration using CuPy and cuML libraries, resulting in `GRACE-GPU` for comparison. We also compare with the GPU-based implementation of the Spectral Modularity Maximization [89] clustering method dubbed as `SMM-GPU`, which operates on the graph adjacency matrix for AGC (or clique expansion of the hypergraph for AHC, or the sum of multiplex adjacency matrices for AMGC) with the attribute KNN augmentation. The results for AHC, AGC, and AMGC are presented in Tables 3.11-3.13, respectively. On the first six smaller datasets in Table 3.11 for AHC, `SMM-GPU` exhibits lower quality in terms of Acc, F1, NMI, and ARI, despite comparable efficiency to `ANCKA-GPU`, which delivers significantly better clustering quality. `ANCKA-GPU` outperforms `GRACE-GPU` in both quality and efficiency

Table 3.11: Additional GPU baselines for AHC.

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

Table 3.12: Additional GPU baselines for AGC.

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

Table 3.13: Additional GPU baselines for AMGC.

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

across all AHC datasets. Notably, on large datasets Amazon and MAG-PM in Table 3.11, `ANCKA-GPU` efficiently produces satisfactory clusters, whereas `GRACE-GPU` and `SMM-GPU` encounter out-of-memory due to their requirement to expand hypergraphs into graphs. Similar observations are made for AGC and AMGC in Tables 3.12 and 3.13. Similar patterns are observed for AGC and AMGC in Tables 3.12 and 3.13. In these tasks, `ANCKA-GPU` delivers superior clustering quality and efficiency on most datasets, except IMDB where ANCKA-GPU is the second best, while SMM-GPU yields lower-quality outcomes and GRACE-GPU falls behind our method in speed. We conclude that ANCKA-GPU offers high clustering quality with remarkable efficiency.

3.8.3 Experimental Analysis

Varying K. Figure 3.5 depicts the Acc, F1, NMI scores, and the KNN computation time of ANCKA on 8 attributed hypergraphs (AHC) when varying K from 2 to 1000. We can make the following observations. First, on most hypergraphs, the clustering accuracies of ANCKA first grow when K is increased from 2 to 10 and then decline, especially when K is beyond 50. The reasons are as follows. When K is small, the KNN graph $\mathcal{G}_K$ in ANCKA fails to capture the key information in the attribute matrix $\mathbf{X}$, leading to limited result quality. On the other hand, when K is large, more noisy or distorted information will be introduced in $\mathcal{G}_K$, and hence, causes accuracy loss. This coincides with our observation in the preliminary study in Figure 3.2. Moreover, as K goes up, the time of KNN construction increases on all datasets. Figures 3.7 and 3.8 show the Acc, F1, NMI scores and KNN computation time of ANCKA on the 6 attributed graphs and 3 attributed multiplex graphs for AGC and AMGC, respectively, when varying K from 2 to 1000. On small graphs in Figure 3.7a-3.7d and Figure 3.8, the cluster quality increases from 2 to 50, and then declines on datasets such as Citeseer-UG, Wiki, and ACM. On large datasets TWeibo and Amazon2M in Figure 3.7e and 3.7f, a turning point appears around $K = 10$. Therefore, we set K to be 50 and 10 on these small and large datasets, respectively.

Varying β. Recall that in the generalized (α, β, γ)-random walk model, the parameter β is used to balance the combination of topological proximities from graph topology $\mathcal{N}_O$ and the attribute similarities from KNN graph $\mathcal{G}_K$. Figure 3.6 displays the AHC performance of ANCKA on 8 attributed hypergraph datasets when β varies from 0 to 1. When $\beta = 0$, ANCKA degrades to a hypergraph clustering method without

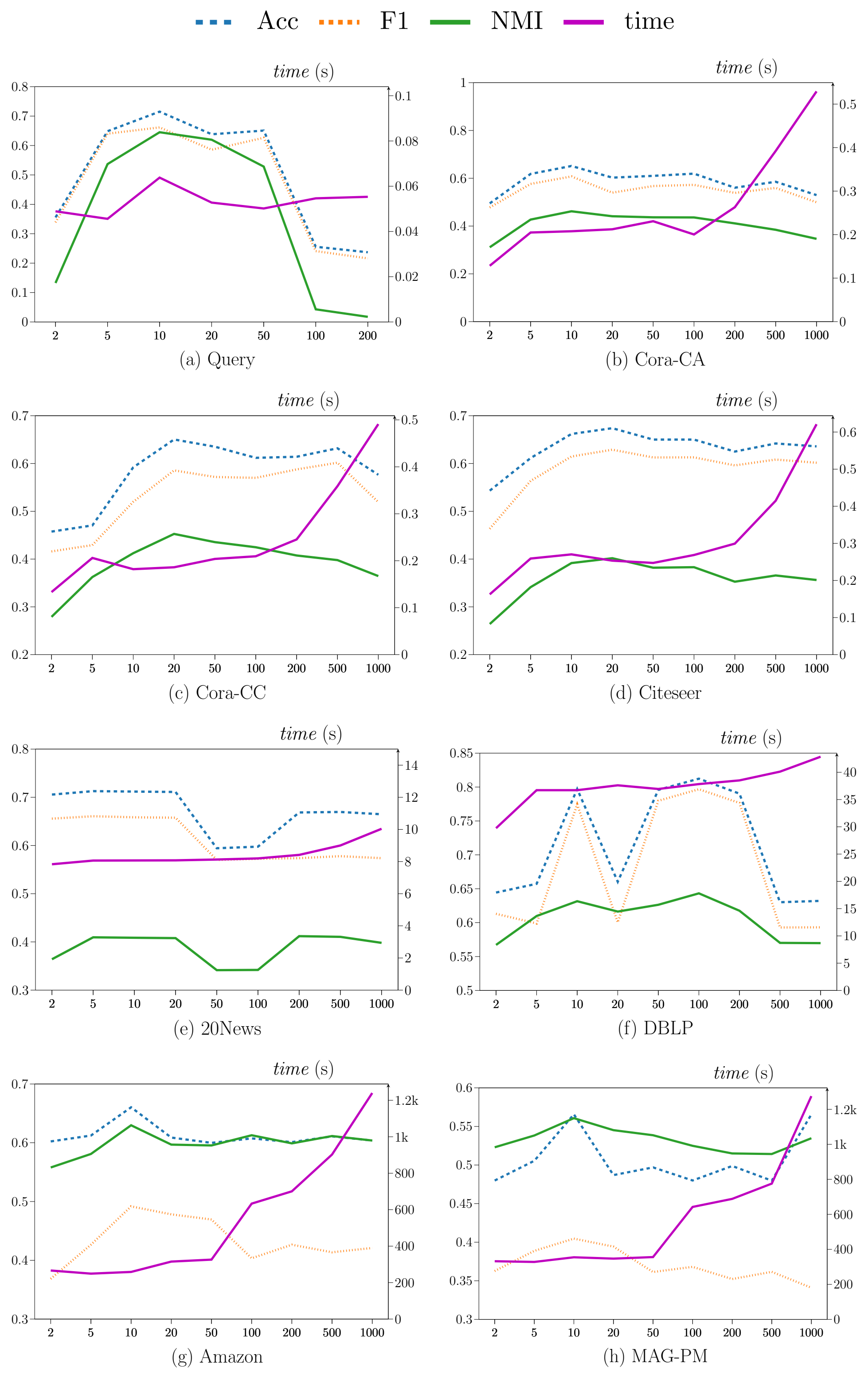


Figure 3.5: Varying K for AHC (best viewed in color).

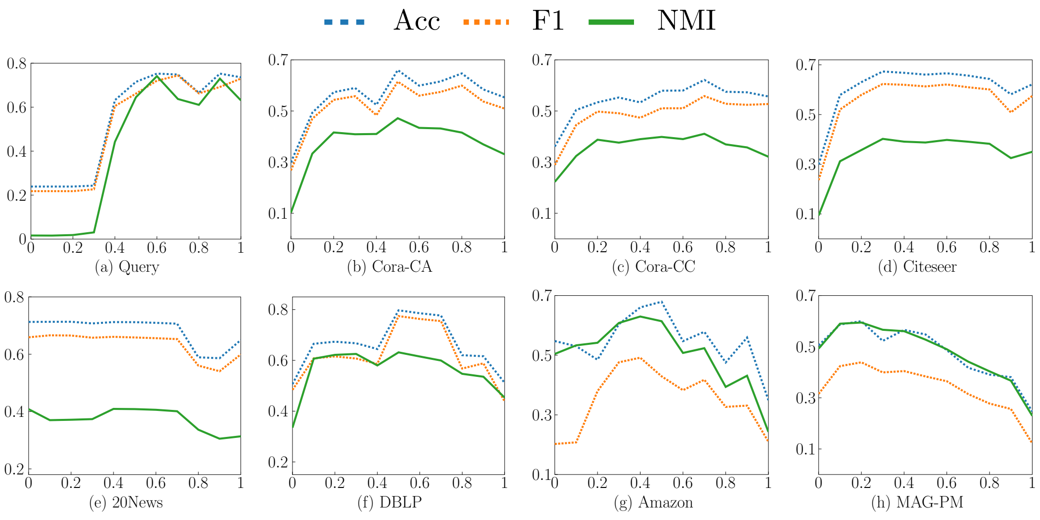


Figure 3.6: Varying β for AHC (best viewed in color).

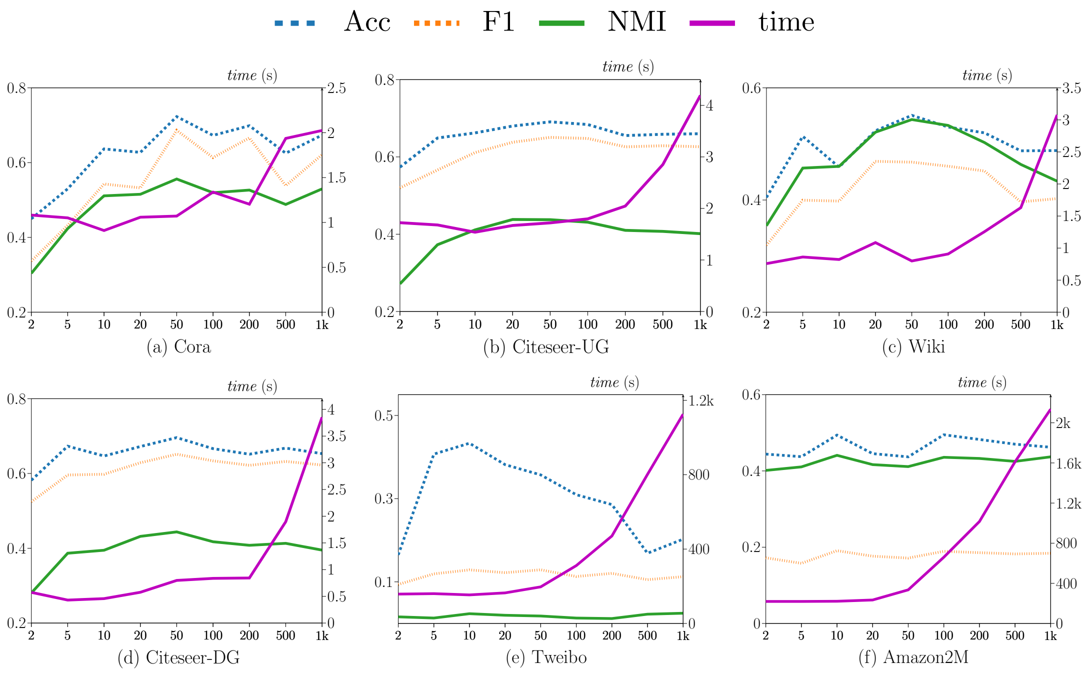


Figure 3.7: Varying K for AGC (best viewed in color).

the consideration of any attribute information, whereas ANCKA only clusters the KNN graph $\mathcal{G}_K$ regardless of the topology structure in $\mathcal{H}$ if $\beta = 1$. From Figure 3.6, we can see a large β (e.g., 0.7-0.8) on small/medium datasets (Query, Cora-CA, Cora-CC, Citeseer, 20News, and DBLP) bring more performance enhancements, meaning that

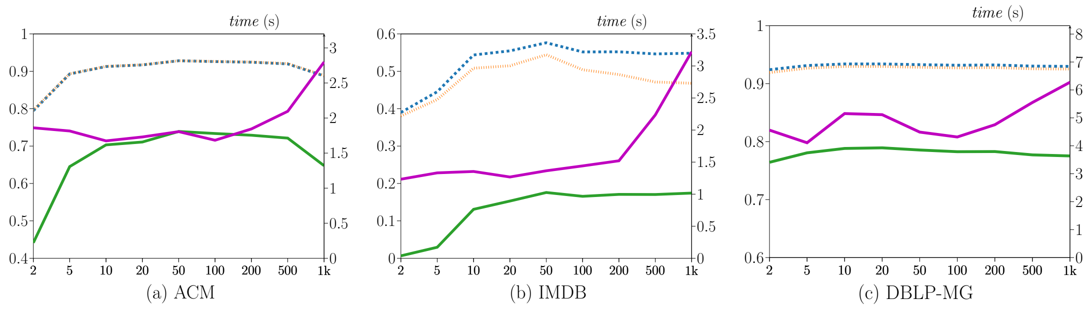


Figure 3.8: Varying K for AMGC (best viewed in color).

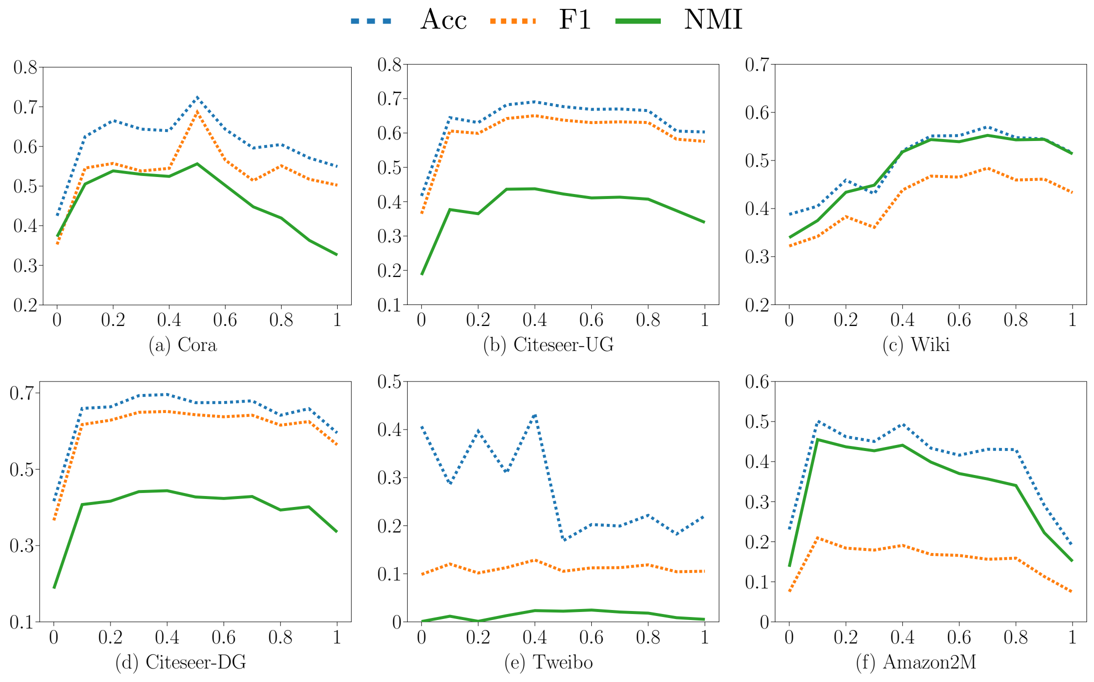


Figure 3.9: Varying β for AGC (best viewed in color).

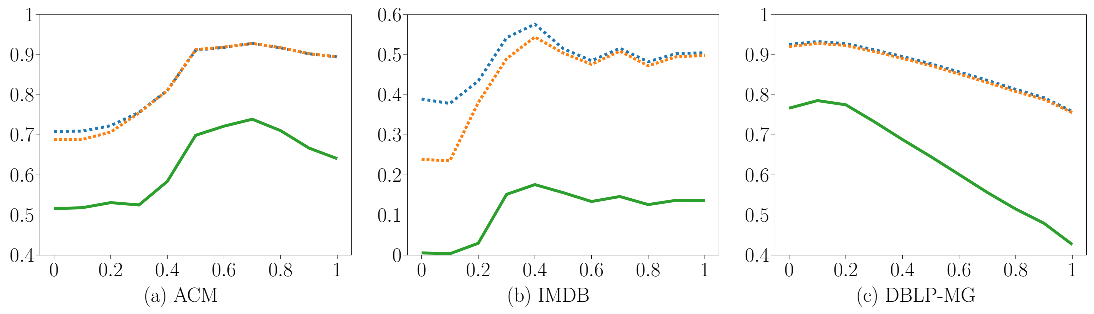


Figure 3.10: Varying β for AMGC (best viewed in color).

Table 3.14: Ablation Analysis on AHC (Time in Seconds).

Algorithm	Query Acc	Query F1	Query NMI	Query Time	Cora-CA Acc	Cora-CA F1	Cora-CA NMI	Cora-CA Time	Cora-CC Acc	Cora-CC F1	Cora-CC NMI	Cora-CC Time	Citeseer Acc	Citeseer F1	Citeseer NMI	Citeseer Time
`ANCKA-random-init`	0.678	**0.662**	0.599	0.393	0.611	0.529	0.438	0.445	0.539	0.493	0.377	0.495	0.567	0.485	0.320	0.694
`ANCKA-k-means`	0.358	0.353	0.148	0.994	0.572	0.478	0.418	0.782	0.571	0.461	0.400	0.933	0.570	0.469	0.338	1.164
`ANCKA`	**0.715**	**0.662**	**0.645**	**0.342**	**0.651**	**0.608**	**0.462**	**0.402**	**0.592**	**0.520**	**0.412**	**0.416**	**0.662**	**0.615**	**0.392**	**0.635**

Algorithm	20News Acc	20News F1	20News NMI	20News Time	DBLP Acc	DBLP F1	DBLP NMI	DBLP Time	Amazon Acc	Amazon F1	Amazon NMI	Amazon Time	MAG-PM Acc	MAG-PM F1	MAG-PM NMI	MAG-PM Time
`ANCKA-random-init`	0.625	0.609	0.361	10.543	0.637	0.603	0.585	**41.00**	0.623	0.297	0.562	1310	0.512	0.396	0.518	**881.7**
`ANCKA-k-means`	0.398	0.360	0.101	9.828	0.652	0.617	0.605	43.11	0.567	0.227	0.558	1492	0.536	0.276	0.504	4437
`ANCKA`	**0.7118**	**0.658**	**0.409**	**8.176**	**0.797**	**0.774**	**0.632**	41.50	**0.660**	**0.492**	**0.630**	**1286**	**0.566**	**0.405**	**0.561**	1371

attribute information plays more important roles on those datasets. This is because they have limited amounts of connections (or are too dense to be informative, e.g., on Query) in the original hypergraph structure as listed in Table 3.1 and rely on attribute similarities from the augmented KNN graph $\mathcal{G}_K$ for improved clustering. By contrast, on Amazon and MAG-PM, `ANCKA` achieves the best clustering quality with small β in $[0.1, 0.4]$, indicating graph topology has higher weights on Amazon and MAG-PM. Figures 3.9 and 3.10 report the Acc, F1, and NMI scores on AGC and AMGC tasks respectively. Similarly, when β increases from 0, the cluster quality generally improves, then becomes stable around 0.4 and 0.5, and decreases when β is large and close to 1. On DBLP-MG in Figure 3.10, the highest clustering quality can be acquired with a small β around 0.1. We infer that node attributes in this dataset are of limited significance for clustering, while on ACM and IMDB, the best quality is achieved when β appropriately balances graph topology and attributes.

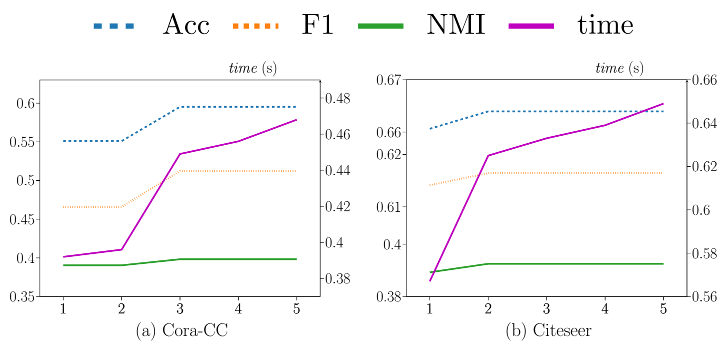


Figure 3.11: Varying γ on Attributed Hypergraphs.

Varying γ. We evaluate `ANCKA` in terms of AHC quality and running time when

Table 3.15: Ablation Analysis on AGC (Time in Seconds).

	Cora				Citeseer-UG				Wiki			
Algorithm	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time
ANCKA-random-init	0.676	0.619	0.544	**0.869**	0.681	**0.681**	0.435	**1.436**	0.507	0.436	0.529	1.082
ANCKA-k-means	0.597	0.456	0.511	1.254	0.684	0.631	**0.440**	1.915	0.459	0.385	0.506	3.030
ANCKA	**0.723**	**0.686**	**0.556**	1.251	**0.691**	0.651	0.438	1.587	**0.551**	**0.467**	**0.543**	**0.907**

	Citeseer-DG				Tweibo				Amazon2M			
Algorithm	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time
ANCKA-random-init	0.689	0.649	0.443	**0.685**	0.364	0.094	0.005	**1068**	0.452	0.188	0.405	**1269**
ANCKA-k-means	0.689	0.636	**0.445**	2.005	0.428	0.067	0.000	1837	0.429	0.107	0.406	3420
ANCKA	**0.696**	**0.651**	0.444	0.838	**0.433**	**0.129**	**0.023**	1318	**0.494**	**0.191**	**0.441**	1708

Table 3.16: Ablation Analysis on AMGC (Time in Seconds).

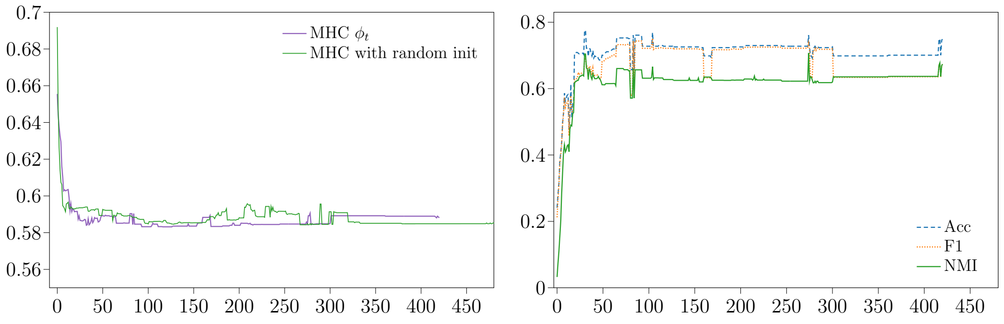
	ACM				IMDB				DBLP-MG			
Algorithm	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time
ANCKA-random-init	0.923	0.924	0.728	**1.546**	0.536	0.482	0.165	**1.131**	0.932	0.928	0.783	4.391
ANCKA-k-means	0.926	0.927	0.738	1.818	0.383	0.203	0.005	1.703	0.926	0.920	0.774	4.719
ANCKA	**0.928**	**0.928**	**0.739**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

varying γ. Figure 3.11 displays the Acc, F1, NMI, and time on two representative datasets when γ varies from 1 to 5. The results on other datasets are similar and thus are omitted for space. Observe that in practice the Acc, F1, and NMI scores obtained by ANCKA first increase and then remain stable when γ is beyond 3 and 2 on Cora-CC and Citeseer, respectively. By contrast, the running time goes up as γ increases. Therefore, we set $\gamma = 3$ in experiments.

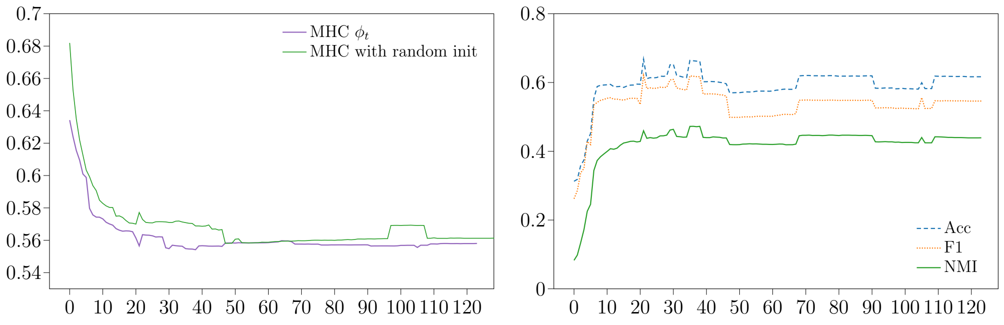
Effectiveness Evaluation of InitBCM and Discretize. On attributed hypergraphs, to verify the effectiveness of InitBCM for the BCM initialization, we compare ANCKA with the ablated version ANCKA-random-init, where the BCM matrix $\mathbf{Y}^{(0)}$ is initialized at random. In Table 3.14, ANCKA obtains remarkable improvements over ANCKA-random-init in Acc, F1, and NMI in comparable processing time. For instance, on Amazon, ANCKA outperforms ANCKA-random-init by a large margin of 3.7% Acc, 19.5% F1, and 6.8% NMI with 24 seconds less to process. On MAG-PM, ANCKA needs additional time compared to ANCKA-random-init. The reason is that ANCKA-random-init starts with a low-quality BCM and converges to local optimum solutions with suboptimal MHC, whereas ANCKA can bypass such pitfalls with a good initial BCM from InitBCM and continue searching for the optimal solution with more iterations, which in turn results in a considerable gap in clustering quality. In addition, we validate the effectiveness of `Discretize` used in `ANCKA` to transform k leading eigenvectors $\mathbf{Q}$ to BCM matrix $\mathbf{Y}$. Table 3.14 reports the accuracy of `ANCKA` and a variant `ANCKA-k-means` obtained by replacing `Discretize` in `ANCKA` with `k-means` on all datasets. It can be observed that compared with `ANCKA-k-means`, `ANCKA` is able to output high-quality BCM matrices $\mathbf{Y}$ with substantially higher clustering accuracy scores while being up to $3.2\times$ faster. The ablation results on AGC and AMGC are in Tables 3.15 and 3.16, respectively. Regarding clustering quality (Acc, F1, NMI), Table 3.15 shows that for AGC, `ANCKA` surpasses its ablated counterparts on all datasets across most effectiveness metrics, except for the Citeseer datasets. For example, `ANCKA` with `InitBCM` achieves an Acc that is 4.2% higher than `ANCKA`-random-init on Amazon2M. In Table 3.16 for AMGC, `ANCKA` performs the best on all the three datasets. For efficiency in Tables 3.15 and 3.16, `ANCKA` is similar to `ANCKA`-random-init, while `ANCKA-k-means` is slower. These results confirm the effectiveness of the proposed techniques for AGC and AMGC.

3.8.4 Convergence Analysis

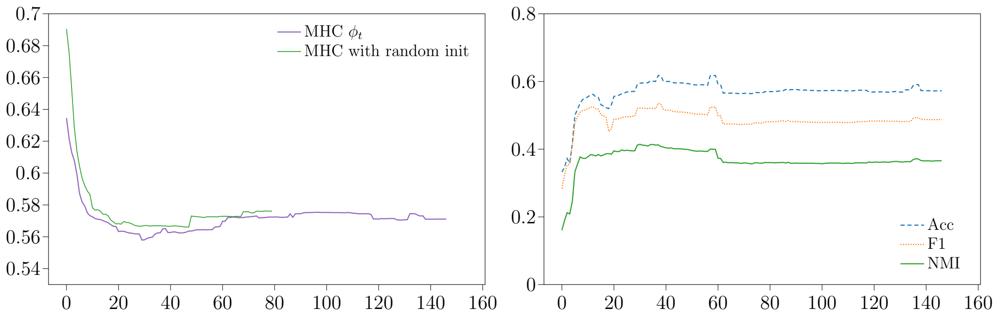
We provide an empirical analysis pertinent to the convergence of `ANCKA` for attributed hypergraph clustering. To do so, we first disable the early termination strategies at Line 10 in Algorithm 1. We also set $\tau = 1$ so as to evaluate the MHC (denoted as ϕ_t) of the BCM matrix $\mathbf{Y}^{(t)}$ generated in each t-th iteration of `ANCKA` and `ANCKA`-random-init, where t starts from 0 till convergence. Furthermore, we calculate the Acc, F1, and NMI scores with the ground truth for each BCM matrix $\mathbf{Y}^{(t)}$ generated throughout the iterative procedures of `ANCKA`. Figures 3.12-3.13 show the MHC ϕ_t, Acc, F1, and NMI scores based on the BCM matrix of each iteration in `ANCKA`, as well as the MHC of `ANCKA`-random-init over all datasets. Notably, MHC ϕ_t experiences a sharp decline when t increases from 0 to 50 on most hypergraphs, while the Acc, F1, and NMI results have significant growth. Moreover, compared to MHC with



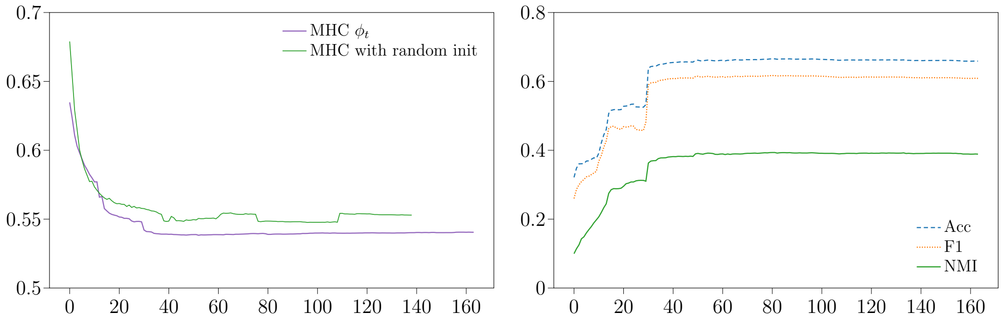
(a) Query



(b) Cora-CA

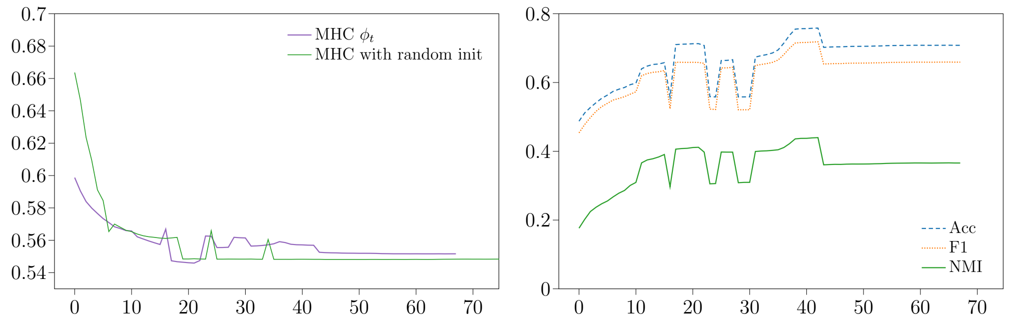


(c) Cora-CC

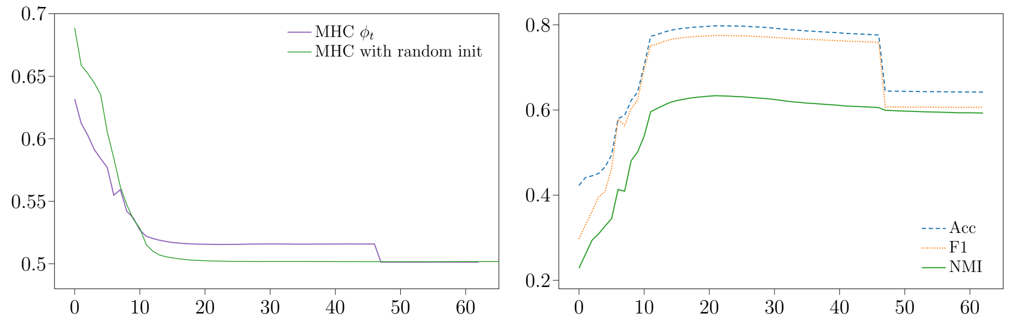


(d) Citeseer

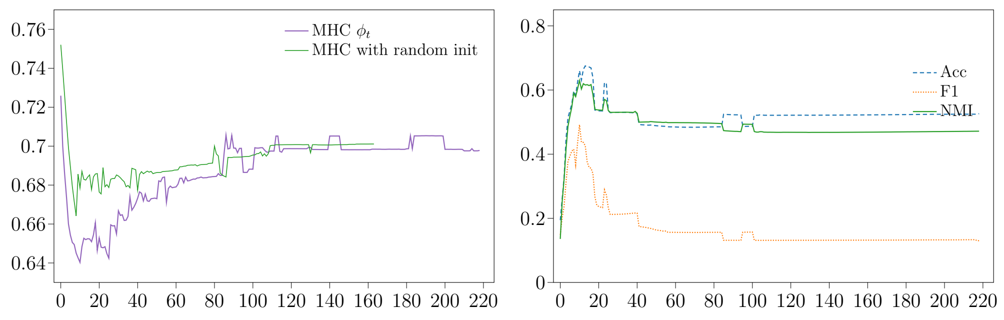
Figure 3.12: Convergence Analysis, part 1 (best viewed in color).



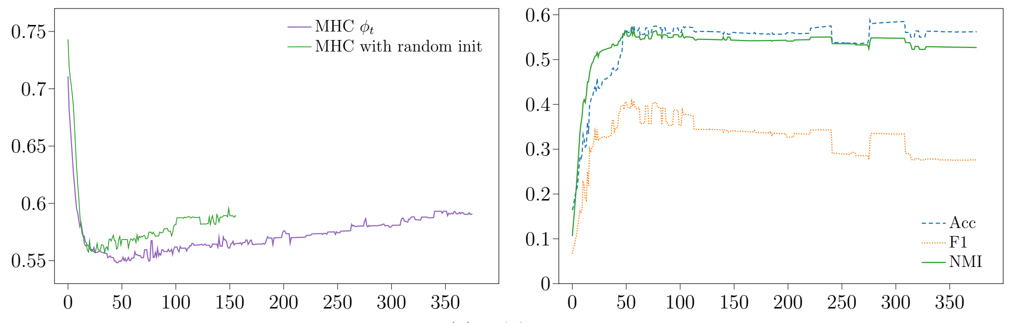
(a) 20News



(b) DBLP



(c) Amazon



(d) MAG-PM

Figure 3.13: Convergence Analysis, part 2 (best viewed in color).

random init, MHC curves of `ANCKA` are mostly lower (better) on all datasets under the same t-th iteration. These phenomena demonstrate the effectiveness of `InitBCM`

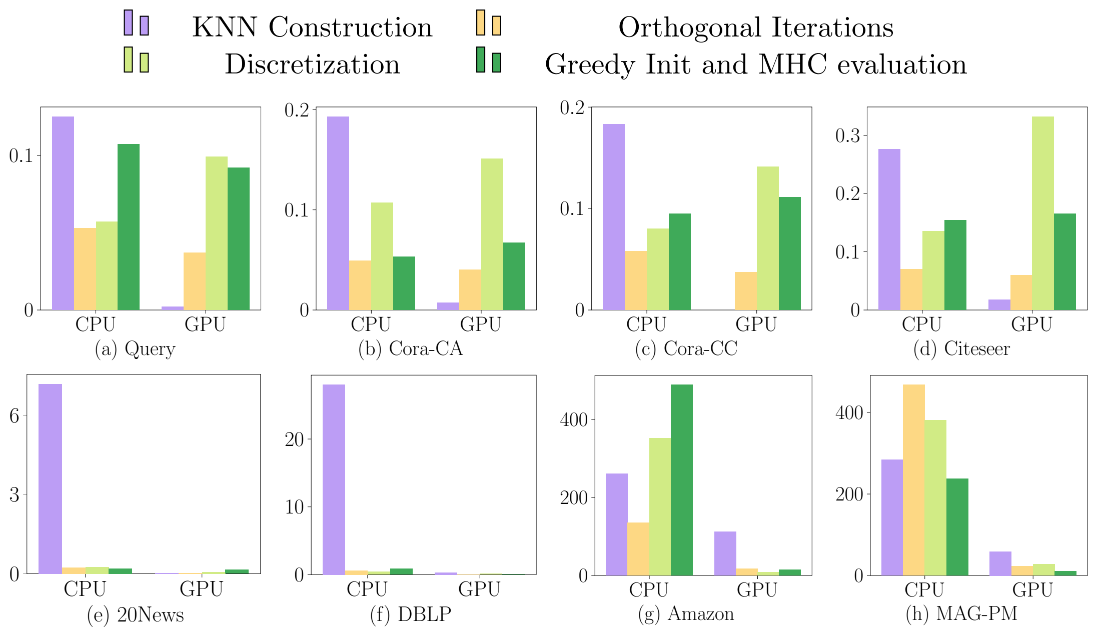


Figure 3.14: Runtime breakdown of CPU-based `ANCKA` and `ANCKA-GPU` in seconds.

in facilitating fast convergence of `ANCKA`. However, when we keep increasing t, these scores either remain stable or deteriorate. For instance, MHC scores grow significantly after 10 iterations on Amazon, while there is a big drop in Acc and F1 scores when $t \geq 45$ on DBLP. This indicates that adding more iterations does not necessarily ensure better solutions. Hence, the early termination proposed in `ANCKA` can serve as an effective approach to remedy this issue.

3.8.5 Runtime Analysis

Figure 3.14 reports time breakdown of `ANCKA` and `ANCKA-GPU` into four parts: KNN construction, orthogonal iterations, discretization, and greedy initialization and MHC evaluation on all attributed hypergraphs. We first explain the results of `ANCKA` on CPUs. On all datasets, the four parts in `ANCKA` all take considerable time to process, except 20News and DBLP, where KNN construction dominates, since 20News and DBLP contain many nodes but relatively few edges. Then, we compare the time breakdown of `ANCKA-GPU` with `ANCKA`. On small attributed hypergraphs (Query, Cora-

Table 3.17: Varying similarity measures in KNN construction for `ANCKA` on AHC.

	Query				Cora-CA				Cora-CC				Citeseer				20News				DBLP				**Quality**
Measure	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	**Rank**
Manhattan	0.595	0.558	0.537	0.386	0.491	0.432	0.304	0.247	0.470	0.384	0.243	0.185	0.286	0.192	0.109	0.022	0.596	0.571	0.337	0.330	0.522	0.453	0.351	0.284	4.5
Euclidean	0.669	0.631	0.619	0.484	0.469	0.411	0.277	0.212	0.471	0.384	0.244	0.186	0.303	0.286	0.109	0.060	0.658	0.568	0.396	0.427	0.525	0.489	0.350	0.285	4.0
Sigmoid	0.316	0.323	0.084	0.041	0.572	0.497	0.415	0.341	**0.687**	**0.648**	**0.460**	**0.424**	0.653	0.606	0.387	0.393	**0.712**	**0.661**	0.394	0.456	0.703	0.714	0.563	0.486	2.5
Angular	**0.721**	**0.667**	**0.659**	**0.576**	0.504	0.463	0.397	0.302	0.576	0.508	0.391	0.323	0.650	0.599	0.377	0.387	**0.712**	0.658	**0.410**	**0.470**	0.662	0.602	0.617	0.556	2.3
Cosine	0.715	0.662	0.645	0.571	**0.651**	**0.608**	**0.462**	**0.406**	0.592	0.520	0.412	0.338	**0.662**	**0.615**	**0.392**	**0.397**	**0.712**	0.658	0.409	0.469	**0.797**	**0.774**	**0.632**	**0.632**	**1.5**

Table 3.18: Varying similarity measures in KNN construction for `ANCKA` on AGC.

	Cora				Citeseer-UG				Wiki				Citeseer-DG				**Quality**
Measure	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	**Rank**
Manhattan	0.545	0.473	0.446	0.321	0.472	0.432	0.234	0.233	0.416	0.318	0.312	0.199	0.583	0.540	0.306	0.304	4.5
Euclidean	0.569	0.486	0.453	0.353	0.408	0.380	0.182	0.161	0.440	0.361	0.370	0.228	0.665	0.591	0.395	0.410	4.0
Sigmoid	0.659	0.562	0.518	0.443	0.684	0.648	0.431	0.441	0.537	0.459	0.510	0.343	0.691	0.649	0.437	0.450	2.1
Angular	0.651	0.557	0.528	0.447	0.674	0.636	0.424	0.434	0.367	0.296	0.308	0.185	0.687	0.645	0.434	0.447	3.4
Cosine	**0.723**	**0.686**	**0.556**	**0.484**	**0.691**	**0.651**	**0.438**	**0.450**	**0.551**	**0.467**	**0.543**	**0.353**	**0.696**	**0.651**	**0.444**	**0.460**	**1.0**

CA, Cora-CC, and Citeseer) in Figures 3.14a, 3.14b, 3.14c, and 3.14d, observe that `ANCKA-GPU` significantly reduces the time for KNN, while the other time costs are on par with that of `ANCKA`, which is consistent with the results in Section 3.8.2. On medium-sized/large attributed hypergraphs in Figures 3.14e, 3.14f, 3.14g, and 3.14h, `ANCKA-GPU` significantly improves the efficiency on all of KNN construction, orthogonal iterations, discretization, greedy initialization and MHC evaluation. From the results on Amazon and MAG-PM, we observe that the scalability of `ANCKA-GPU` is primarily constrained by KNN construction, while the overhead of the CPU-based `ANCKA` is more evenly distributed across the four parts.

Table 3.19: Varying similarity measures in KNN construction for ANCKA on AMGC.

	ACM				IMDB				DBLP-MG				**Quality**
Measure	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	**Rank**
Manhattan	0.748	0.754	0.459	0.446	0.375	0.372	0.006	0.006	**0.933**	**0.929**	**0.785**	0.838	3.4
Euclidean	0.749	0.755	0.460	0.448	0.361	0.356	0.003	0.002	**0.933**	**0.929**	**0.785**	0.838	3.4
Sigmoid	0.922	0.923	0.717	0.781	0.559	0.528	0.164	0.187	0.932	0.928	0.783	0.837	3.0
Angular	0.927	0.927	0.736	0.793	0.530	0.495	0.161	0.177	0.927	0.922	0.770	0.825	3.3
Cosine	**0.928**	**0.928**	**0.739**	**0.796**	**0.576**	**0.544**	**0.176**	**0.195**	**0.933**	**0.929**	**0.785**	**0.839**	**1.0**

3.8.6 Additional Experiments

Effect of Various Attribute Similarity Measures

We conduct additional experiments on various similarity measures for KNN construction. For two d-dimensional attribute vectors x and y, in addition to Cosine similarity $Cosine(x, y)$ utilized in ANCKA, we assess normalized Manhattan and Euclidean distances transformed by a Laplacian kernel into the range $[0, 1]$ as similarity scores, Sigmoid similarity represented by a hyperbolic tangent kernel function, and Angular similarity normalized to $[0, 1]$ from angular distance. The formulas for these measures are provided in Eq. (3.30). Laplacian kernels are commonly employed to normalize Manhattan and Euclidean distances for affinity matrix construction, with γ set to d^{-1} by default [151]. The performance of ANCKA using each similarity measure for attributed KNN construction in AHC, AGC, and AMGC is detailed in Tables 3.17, 3.18, and 3.19, respectively. Similarity measures are sorted by each metric, and we list their average Quality Rank for each task in the last column. Notably, across the three attributed network clustering tasks—AHC, AGC, and AMGC—ANCKA equipped with Cosine similarity secures the best quality rank (1.5, 1.0, and 1.0, respectively) across all evaluation metrics and datasets. Meanwhile, ANCKA with either Sigmoid or

Table 3.20: Objective values $f(\mathcal{C})$ achieved by a method and $f(\mathcal{C}^*)$ of ground truth for AHC. The better value of each objective is highlighted.

	Query					Cora-CC					Cora-CA					Citeseer				
Algorithm	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$
`Infomap`	0.235	0.215	0.017	6.704	8.976	0.514	0.464	0.343	8.121	10.36	0.541	0.479	0.393	9.298	10.26	0.491	0.463	0.263	8.890	10.58
`Louvain`	0.239	0.218	0.017	0.825	0.501	0.501	0.430	0.332	0.817	0.680	0.569	**0.546**	0.373	0.735	0.667	0.570	0.486	0.319	0.787	0.641
`ANCKA`	**0.715**	**0.662**	**0.645**	0.583	0.585	**0.651**	**0.608**	**0.462**	0.555	0.583	**0.592**	0.520	**0.412**	0.558	0.594	**0.662**	**0.615**	**0.392**	0.539	0.595
	20News					DBLP					Amazon					MAG-PM				
Algorithm	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$
`Infomap`	0.338	0.129	0.004	13.56	14.18	0.595	0.573	0.448	10.50	13.47	OOM					0.398	0.172	0.380	12.03	14.39
`Louvain`	0.633	0.522	0.304	0.260	0.198	0.643	0.580	0.554	0.856	0.788	OOM					OOM				
`ANCKA`	**0.712**	**0.658**	**0.409**	0.546	0.594	**0.797**	**0.774**	**0.632**	0.516	0.524	**0.660**	**0.492**	**0.630**	0.640	0.641	**0.566**	**0.405**	**0.561**	0.549	0.595

Table 3.21: Objective values $f(\mathcal{C})$ achieved by a method and $f(\mathcal{C}^*)$ of ground truth for AGC. The better value of each objective is highlighted.

	Cora					Citeseer-UG					Wiki				
Algorithm	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$
`Infomap`	0.569	0.503	0.455	10.87	11.30	0.590	0.546	0.312	9.232	11.04	0.467	0.417	0.468	7.669	9.175
`Louvain`	0.671	0.640	0.474	0.472	0.374	0.680	0.621	0.426	0.488	0.369	**0.611**	**0.513**	**0.572**	0.668	0.530
`ANCKA`	**0.723**	**0.686**	**0.556**	0.555	0.581	**0.696**	**0.651**	**0.444**	0.539	0.593	0.551	0.467	0.543	0.590	0.643
	Citeseer-DG					Tweibo					Amazon2M				
Algorithm	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$
`Infomap`	0.621	0.565	0.357	9.206	11.04	0.417	0.084	0.000	16.89	18.90	0.357	0.191	0.424	15.26	18.60
`Louvain`	0.682	0.617	0.419	0.475	0.359	0.271	0.113	0.015	0.383	0.252	0.463	0.154	0.429	0.790	0.669
`ANCKA`	**0.696**	**0.651**	**0.444**	0.539	0.593	**0.433**	**0.129**	**0.023**	0.723	0.758	**0.494**	**0.191**	**0.441**	0.612	0.689

Angular similarities exhibits the second-best performance.

$$
\begin{aligned}
&Cosine(x,y) = \frac{xy^\top}{\|x\|\|y\|}\\
&Manhattan(x,y) = e^{-\gamma\|x-y\|_1}\\
&Euclidean(x,y) = e^{-\gamma\|x-y\|_2}\\
&Angular(x,y) = 1 - \frac{\cos^{-1} Cosine(x,y)}{\pi}\\
&Sigmoid(x,y) = \tanh(\gamma xy^\top + 1)
\end{aligned}
\tag{3.30}
$$

Comparing Clustering Objectives

In `ANCKA`, we formulate the multi-hop conductance (MHC) objective as $\Phi(\mathbf{Y}) = 1 - \Psi(\mathbf{Y})$, where $\mathbf{Y}$ is a cluster membership matrix returned by a method. Our goal is to minimize Φ, which is equivalent to maximizing $\Psi(\mathbf{Y})$. For ground truth clusters $\mathbf{Y}^*$,

Table 3.22: Objective values $f(\mathcal{C})$ achieved by a method and $f(\mathcal{C}^*)$ of ground truth for AMGC. The better value of each objective is highlighted.

	ACM					IMDB					DBLP-MG				
Algorithm	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$	Acc	F1	NMI	$f(\mathcal{C})$	$f(\mathcal{C}^*)$
`Infomap`	0.653	0.665	0.418	10.01	11.05	0.412	0.362	0.027	11.32	12.03	0.296	0.116	0.002	11.91	12.60
`Louvain`	0.659	0.670	0.422	0.555	0.356	0.452	0.392	0.057	0.460	0.348	0.909	0.900	0.731	0.172	0.161
`ANCKA`	**0.928**	**0.928**	**0.739**	0.563	0.567	**0.576**	**0.544**	**0.176**	0.584	0.599	**0.933**	**0.929**	**0.785**	0.590	0.595

the fact that $\Psi(\mathbf{Y}^*) < \Psi(\mathbf{Y})$, i.e., $\Phi(\mathbf{Y}^*) > \Phi(\mathbf{Y})$, indicates that `ANCKA` can identify a clustering with lower MHC than the ground truth clusters.

This phenomenon is not unique to our method but is also common in other clustering methods with objectives, such as the map equation and modularity. To compare these objective functions, we use KNN augmented graphs to test `Infomap`, which minimizes the map equation objective, and `Louvain`, which maximizes modularity. We then calculate the objective values of the clusters returned by these methods and compare them to the objective values of the ground truth. The results are presented in Tables 3.20-3.22, where $f(\mathcal{C})$ denotes the objective value of the clustering result returned by a method, and f represents the multi-hop conductance, modularity, and map equation objectives for `ANCKA`, `Louvain`, and `Infomap`, respectively. The value $f(\mathcal{C}^*)$ corresponds to the objective value of the ground truth. In Tables 3.20-3.22, all three algorithms yield clusters with a more favorable objective value $f(\mathcal{C})$ than that of the ground truth clusters $f(\mathcal{C}^*)$. Nevertheless, the fact that `ANCKA` consistently achieves the best clustering quality showcases the effectiveness of the MHC objective.

Comparing the Clusters of `ANCKA` and `ANCKA-GPU`

We evaluate the NMI and ARI metrics between the clusters obtained by `ANCKA` (`C`*) and the clusters obtained by `ANCKA-GPU` (`G`*). The results are shown in Table 3.23, where the columns labeled `C`*-`G`* display the NMI and ARI scores between `C`* and `G`*, whereas the remaining columns show the NMI and ARI scores between the ground truth `GT` and either `C`* or `G`*. It is noteworthy that the NMI and ARI scores for

Table 3.23: Evaluation between CPU-based ANCKA (C*) and ANCKA-GPU (G*). GT stands for ground truth clusters.

Task	Dataset	NMI			ARI		
		C*-GT	G*-GT	C*-G*	C*-GT	G*-GT	C*-G*
AHC	Query	0.645	0.666	0.780	0.571	0.578	0.732
	Cora-CA	0.462	0.469	0.852	0.406	0.411	0.878
	Cora-CC	0.412	0.395	0.675	0.338	0.311	0.623
	Citeseer	0.392	0.387	0.750	0.397	0.410	0.767
	20News	0.409	0.407	0.835	0.469	0.465	0.876
	DBLP	0.632	0.643	0.920	0.632	0.646	0.943
	Amazon	0.630	0.636	0.750	0.524	0.509	0.763
	MAG-PM	0.561	0.545	0.743	0.471	0.454	0.757
AGC	Cora	0.556	0.533	0.765	0.484	0.470	0.773
	Citeseer-UG	0.438	0.437	0.922	0.450	0.451	0.940
	Wiki	0.543	0.547	0.847	0.353	0.368	0.724
	Citeseer-DG	0.444	0.441	0.863	0.460	0.454	0.892
	TWeibo	0.023	0.022	0.115	0.019	0.016	0.176
	Amazon2M	0.441	0.437	0.609	0.545	0.544	0.698
AMGC	ACM	0.739	0.730	0.953	0.796	0.786	0.974
	IMDB	0.176	0.166	0.444	0.195	0.184	0.525
	DBLP-MG	0.785	0.791	0.977	0.839	0.842	0.988

C*-G* are consistently higher than those relative to the ground truth. This indicates a strong agreement between the clustering results of ANCKA and ANCKA-GPU.

Effect of similarity threshold in KNN construction

Previous research [84] suggests that KNN edges with low similarity should be avoided. We explore two ways for applying a similarity threshold to prune edges in the constructed KNN graph. After constructing the KNN graph, we can either use an absolute similarity threshold ϵ_s to remove edges with similarity scores below ϵ_s, or employ a relative similarity threshold ϵ_p, expressed as a percentage, to eliminate the bottom ϵ_p% of edges. A limitation of ϵ_s is that similarity score distributions can vary across datasets, and as a result, a threshold ϵ_s that is effective for one dataset might be insufficient for another. Therefore, we also test setting ϵ_p as an alternative to ϵ_s. We vary ϵ_s from 0.1 to 0.3 and ϵ_p from 10% to 30%, and report the results for AHC in Table 3.24. All tested settings are sorted by each metric, and we list their

Table 3.24: Impact of exerting absolute or relative similarity thresholds (ϵ_s or ϵ_p) on the KNN graph on AHC.

	Query				Cora-CA				Cora-CC			
Setting	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI
$\epsilon_s = 0.1$	**0.715**	**0.662**	**0.645**	**0.571**	**0.661**	**0.616**	**0.473**	**0.422**	0.590	0.518	**0.417**	**0.339**
$\epsilon_s = 0.2$	**0.715**	**0.662**	**0.645**	**0.571**	0.655	0.612	0.472	0.417	0.572	0.495	0.370	0.321
$\epsilon_s = 0.3$	**0.715**	**0.662**	**0.645**	**0.571**	0.530	0.498	0.324	0.246	0.438	0.378	0.303	0.233
$\epsilon_p = 10$	0.709	0.637	0.611	0.549	0.594	0.524	0.431	0.364	0.553	0.465	0.377	0.329
$\epsilon_p = 20$	0.511	0.509	0.346	0.256	0.631	0.586	0.434	0.385	0.511	0.483	0.363	0.270
$\epsilon_p = 30$	0.422	0.395	0.237	0.182	0.615	0.563	0.409	0.371	0.532	0.450	0.353	0.285
ANCKA	**0.715**	**0.662**	**0.645**	**0.571**	0.651	0.608	0.462	0.406	**0.592**	**0.520**	0.412	0.338

	Citeseer				20News				DBLP				**Quality**
Setting	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	**Rank**
$\epsilon_s = 0.1$	0.654	0.606	0.382	0.390	**0.712**	0.658	**0.409**	**0.469**	0.797	0.774	0.632	0.632	**1.7**
$\epsilon_s = 0.2$	0.537	0.463	0.287	0.287	**0.712**	0.658	**0.409**	**0.469**	0.804	0.784	**0.641**	0.639	2.7
$\epsilon_s = 0.3$	0.332	0.291	0.121	0.076	**0.712**	0.658	**0.409**	**0.469**	0.552	0.456	0.468	0.361	5.2
$\epsilon_p = 10$	0.606	0.563	0.334	0.326	0.710	**0.660**	0.368	0.417	0.650	0.591	0.603	0.548	4.8
$\epsilon_p = 20$	0.655	0.568	0.369	0.365	0.711	0.656	0.407	0.467	**0.816**	**0.800**	0.640	**0.645**	4.0
$\epsilon_p = 30$	0.586	0.538	0.295	0.299	0.662	0.566	0.404	0.433	0.642	0.609	0.574	0.507	5.8
ANCKA	**0.662**	**0.615**	**0.392**	**0.397**	**0.712**	0.658	**0.409**	**0.469**	0.797	0.774	0.632	0.632	1.8

Table 3.25: Impact of exerting absolute or relative similarity thresholds (ϵ_s or ϵ_p) on the KNN graph on AGC.

	Cora				Citeseer-UG				Wiki				Citeseer-DG				**Quality**
Setting	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	**Rank**
$\epsilon_s = 0.1$	0.644	0.549	0.526	0.454	0.684	0.646	0.434	0.446	0.539	0.460	0.527	0.339	0.670	0.637	**0.446**	0.453	3.9
$\epsilon_s = 0.2$	0.517	0.421	0.477	0.349	0.685	0.604	0.409	0.427	**0.563**	0.448	0.524	**0.377**	0.687	0.592	0.413	0.423	5.0
$\epsilon_s = 0.3$	0.494	0.410	0.402	0.289	0.427	0.362	0.178	0.155	0.459	0.384	0.411	0.257	0.469	0.391	0.206	0.188	7.0
$\epsilon_p = 10$	0.686	0.582	0.537	0.469	0.700	**0.653**	0.448	**0.464**	0.522	0.423	0.528	0.327	0.659	0.628	0.437	0.443	3.4
$\epsilon_p = 20$	0.676	0.576	0.530	0.465	0.653	0.616	0.431	0.437	0.558	0.451	0.527	0.365	**0.704**	0.646	0.445	**0.460**	3.3
$\epsilon_p = 30$	0.674	0.565	0.522	0.480	**0.714**	0.619	**0.451**	**0.464**	**0.563**	0.433	0.527	0.376	0.703	0.635	0.429	0.452	3.0
ANCKA	**0.723**	**0.686**	**0.556**	**0.484**	0.691	0.651	0.438	0.450	0.551	**0.467**	**0.543**	0.353	0.696	**0.651**	0.444	**0.460**	**2.1**

Table 3.26: Impact of exerting absolute or relative similarity thresholds (ϵ_s or ϵ_p) on the KNN graph on AMGC.

	ACM				IMDB				DBLP-MG				**Quality**
Setting	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	**Rank**
$\epsilon_s = 0.1$	0.927	0.927	0.737	0.794	0.575	0.552	0.169	0.188	**0.933**	0.928	0.786	0.838	2.2
$\epsilon_s = 0.2$	0.927	0.927	0.737	0.793	0.388	0.230	0.005	0.003	**0.933**	**0.929**	0.786	0.838	3.4
$\epsilon_s = 0.3$	0.921	0.921	0.720	0.779	0.386	0.253	0.002	0.001	**0.933**	**0.929**	**0.787**	**0.839**	3.9
$\epsilon_p = 10$	0.920	0.920	0.716	0.776	0.569	**0.557**	0.155	0.175	0.932	0.928	0.785	0.837	4.1
$\epsilon_p = 20$	0.919	0.919	0.713	0.774	0.532	0.518	0.153	0.165	**0.933**	0.928	0.785	0.837	4.7
$\epsilon_p = 30$	0.917	0.917	0.708	0.768	0.556	0.536	0.145	0.162	0.932	0.928	0.784	0.836	5.8
ANCKA	**0.928**	**0.928**	**0.739**	**0.796**	**0.576**	0.544	**0.176**	**0.195**	**0.933**	**0.929**	0.785	**0.839**	**1.4**

average Quality Rank in the last column. From these experiment results, three observations emerge. First, ANCKA delivers the best results on four out of six datasets and is the second-best on the remaining two. Second, as ϵ_s increases, the clustering quality deteriorates on Cora-CA, Cora-CC, and Citeseer; remains unchanged on

Table 3.27: Impact of conflicting attribute and network information on AHC performance of `ANCKA`.

	Query				Cora-CA				Cora-CC			
Setting	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI
`Shuffled-`**X**	0.214	0.207	0.011	0.002	0.191	0.172	0.018	0.003	0.182	0.165	0.011	0.003
`Original`	**0.715**	**0.662**	**0.645**	**0.571**	**0.651**	**0.608**	**0.462**	**0.406**	**0.592**	**0.520**	**0.412**	**0.338**
	Citeseer				20News				DBLP			
Setting	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI
`Shuffled-`**X**	0.187	0.183	0.005	0.001	0.302	0.290	0.008	0.006	0.198	0.178	0.002	0.001
`Original`	**0.662**	**0.615**	**0.392**	**0.397**	**0.712**	**0.658**	**0.409**	**0.469**	**0.797**	**0.774**	**0.632**	**0.632**

Table 3.28: Impact of conflicting attribute and network information on AGC performance of `ANCKA`.

	Cora				Citeseer-UG				Wiki				Citeseer-DG			
Setting	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI
`Shuffled-`**X**	0.395	0.343	0.271	0.183	0.418	0.360	0.181	0.141	0.188	0.179	0.119	0042	0.316	0.290	0.074	0.073
`Original`	**0.723**	**0.686**	**0.556**	**0.484**	**0.691**	**0.651**	**0.43**8	**0.450**	**0.551**	**0.467**	**0.543**	**0.353**	**0.696**	**0.651**	**0.444**	**0.460**

Table 3.29: Impact of conflicting attribute and network information on AMGC performance of `ANCKA`.

	ACM				IMDB				DBLP-MG			
Setting	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI	Acc	F1	NMI	ARI
`Shuffled-`**X**	0.349	0.347	0.001	0.001	0.374	0.370	0.006	0.006	0.928	0.923	0.770	0.825
`Original`	**0.928**	**0.928**	**0.739**	**0.796**	**0.576**	**0.544**	**0.176**	**0.195**	**0.933**	**0.929**	**0.785**	**0.839**

Query and 20News, suggesting that the ϵ_s values are insufficient for edge filtering; and improves before declining on DBLP. These results do not reveal a consistent relationship between quality and varied ϵ_s across the datasets. Third, with an increasing ϵ_p, performance drops on Query and Cora-CC; improves then declines on Cora-CA, Citeseer, 20News, and DBLP, also indicating inconsistent patterns between quality and varied ϵ_p. Similar trends are observed for AGC and AMGC in Tables 3.25 and 3.26, where `ANCKA` attains the best or second-best performance under most metrics. However, adjusting ϵ_s and ϵ_p does not consistently enhance performance across all datasets.

Table 3.30: Clustering performance comparison between FSSC and ANCKA on AHC (Time in Seconds).

	Query				Cora-CA				Cora-CC				Citeseer			
Algorithm	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time
FSSC	0.601	0.584	0.518	0.183	0.376	0.244	0.272	0.352	0.381	0.248	0.232	0.403	0.386	0.330	0.264	0.500
ANCKA	**0.715**	**0.662**	**0.645**	0.342	**0.651**	**0.608**	**0.462**	0.402	**0.592**	**0.520**	**0.412**	0.416	**0.662**	**0.615**	**0.392**	0.635
	20News				DBLP				Amazon				MAG-PM			
Algorithm	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time
FSSC	0.286	0.267	0.002	6.702	0.419	0.304	0.410	28.49	0.311	0.032	0.000	888.1	0.304	0.050	0.095	601.9
ANCKA	**0.712**	**0.658**	**0.409**	8.176	**0.797**	**0.774**	**0.632**	41.50	**0.660**	**0.492**	**0.630**	1286	**0.566**	**0.405**	**0.561**	1371

Table 3.31: Clustering performance comparison between FSSC and ANCKA on AGC (Time in Seconds).

	Cora				Citeseer-UG				Wiki			
Algorithm	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time
FSSC	0.494	0.391	0.358	0.763	0.301	0.207	0.104	1.151	0.229	0.182	0.198	0.716
ANCKA	**0.723**	**0.686**	**0.556**	1.251	**0.691**	**0.651**	**0.438**	1.587	**0.551**	**0.467**	**0.543**	0.907
	Citeseer-DG				Tweibo				Amazon2M			
Algorithm	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time
FSSC	0.294	0.233	0.099	0.493	0.196	0.103	0.004	761.3	0.185	0.044	0.146	551.9
ANCKA	**0.696**	**0.651**	**0.444**	0.838	**0.433**	**0.129**	**0.023**	1318	**0.494**	**0.191**	**0.441**	1708

Table 3.32: Clustering performance comparison between FSSC and ANCKA on AMGC (Time in Seconds).

	ACM				IMDB				DBLP-MG			
Algorithm	Acc	F1	NMI	Time	Acc	F1	NMI	Time	Acc	F1	NMI	Time
FSSC	0.754	0.750	0.537	1.566	0.396	0.362	0.010	1.156	0.297	0.124	0.003	3.008
ANCKA	**0.928**	**0.928**	**0.739**	1.738	**0.576**	**0.544**	**0.176**	1.574	**0.933**	**0.929**	**0.785**	3.766

Impact of Conflicting Attributes and Networks

We investigate the effect of conflicting network and attribute information. In real-world network data, attribute and network information typically exhibit correlation, aligning with the notion that entities with similar attributes, such as individuals, are more likely to be connected. To artificially induce a conflicting relationship between attributes and network information, we take a given dataset and randomly permute the attribute vectors in the attribute matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, re-assigning them to the n nodes. This process disrupts the natural correlation, thereby creating conflicts between attributes and network information, dubbed as Shuffled-**X**. We then compare the performance of `ANCKA` on both the original data and the Shuffled-**X** data, with results presented in Tables 3.27, 3.28, and 3.29 for the AHC, AGC, and AMGC tasks, respectively. The Shuffled-**X** data consistently yield significantly lower clustering quality compared to `ANCKA` on the original data, confirming that conflicting network and attribute information detrimentally affects clustering quality. This also reaffirms that real datasets typically feature correlated attributes and networks. In the future, we will explore the robustness of clustering against attacks that introduce such conflicts.

Effect of using fewer eigenvectors

The spectral clustering algorithm proposed in [83] leverages the linear space spanned by $O(\log k)$ leading eigenvectors. We adapt this method, denoted by `FSSC`, for the attributed network clustering tasks. Specifically, `FSSC` gets the $\log_2 k$ leading vectors over the transition matrix of the proposed random walk model in the paper, and perform clustering. The results are reported in Table 3.30, 3.31, and 3.32 for AHC, AGC, and AMGC, respectively. `FSSC` yields lower clustering quality on all datasets, making its speedup less significant.

3.9 Summary

This chapter introduces `ANCKA`, a versatile and efficient method for attributed network clustering that supports AHC, AGC, and AMGC tasks. Its superior performance over existing approaches stems from three key innovations: (i) a KNN augmentation strategy that selectively exploits attribute information, (ii) a novel random walk-based problem formulation, and (iii) an iterative optimization framework enhanced with speedup techniques.

To further accelerate computation on large datasets, we develop `ANCKA-GPU`, a GPU-based variant that outperforms its CPU-parallel counterpart while maintaining high clustering quality. Extensive experiments on real-world datasets demonstrate the effectiveness and efficiency of our methods.

Chapter 4

SAHE: Attributed Hypergraph Embedding

This chapter presents `SAHE`, an efficient approach to attributed hypergraph embedding, advancing the thesis's goal of developing scalable solutions for clustering and embedding in attributed network structures. Complementing the attributed hypergraph clustering method in Chapter 3, `SAHE` addresses the challenge of generating high-quality node and hyperedge embeddings for attributed hypergraphs, while also leveraging the KNN augmentation approach to incorporate attribute information. These works, alongside the multi-view integration method in Chapter 5, contribute to advancing clustering and embedding of diverse attributed network structures.

4.1 Introduction

An *attributed hypergraph* captures higher-order relationships among a variable number of nodes through *hyperedges*, and the nodes are often associated with *attribute* information. A hyperedge is a generalized edge that connects more than two nodes. The unique characteristics of attributed hypergraphs have played important roles in various domains by describing the multiway relationships among entities. e.g., social networks [7], genomic expression [28], and online shopping sessions [40]. For example, a group-purchase activity of an item links a group of users together, naturally captured via a hyperedge, and the users carry their own profile attributes.

Network embedding is a fundamental problem in graph analytics, garnering attention from both academia [157] and industry [132], and has been studied on various types of simple graphs with pairwise edge connections, such as homogeneous graphs [141, 98, 117] and attributed graphs [143, 125]. However, network embedding on attributed hypergraphs is still in its early stages, with few native solutions that are efficient and effective in the literature.

Hence, in this work, we focus on the problem of *Attributed Hypergraph Node and hyperEdge Embedding* (AHNEE). Given an attributed hypergraph with n nodes and m hyperedges, AHNEE aims to generate compact embedding vectors for each node and hyperedge. Intuitively, node embeddings capture the hyperedge-featured topological and attribute information surrounding nodes, while hyperedge embeddings inherently captures the connections and attribute semantics of groups of nodes around hyperedges. The embeddings are valuable for downstream tasks: node embeddings facilitate node classification [45] and hyperedge link prediction [147], while hyperedge embeddings support hyperedge classification [137].

It is highly challenging to design native AHNEE solutions, due to the complexity of attributed hypergraphs beyond simple graphs, and the need to simultaneously embed nodes and hyperedges, particularly for large attributed hypergraphs. Effective node and hyperedge embeddings should capture both local and long-ranged information via multi-hop paths formed by hyperedges and nodes. Besides, simply aggregating all node embeddings within a hyperedge to get its hyperedge embedding often results in suboptimal performance. Incorporating these considerations into AHNEE computation requires careful designs to ensure effectiveness and targeted optimizations for efficiency in large attributed hypergraphs.

Existing methods either do not natively support attributed hypergraphs or fail to perform efficiently on massive data. An early study [156] uses the hypergraph Laplacian spectrum for node embeddings, while [45, 44] extend graph-based node embedding to hypergraphs. However, these approaches typically do not consider attribute information or hyperedge embedding generation, with some, like [156], overlooking long-range connectivity. A recent class of studies [122, 64, 60] has developed hypergraph neural networks, which often incur significant computational overhead when applied to large-scale hypergraph data. Another way is converting attributed hypergraphs into bipartite or attributed graphs using star-expansion or clique-expansion, followed by applying graph embedding methods [143, 128]. However, the expansions dilute the representation of higher-order connections in hyperedges and result in dense graphs with high computational costs.

To tackle the challenges, we propose `SAHE`, a <u>S</u>calable <u>A</u>ttributed <u>H</u>ypergraph node and hyperedge <u>E</u>mbedding method that unifies the generation of node embeddings and hypergraph embeddings with high result quality and efficiency, advancing the state of the art for the problem of AHNEE. We accomplish this via comprehensive problem formulations and innovative algorithm designs.

We begin by considering an attribute-extended hypergraph $\mathcal{H}$, which integrates node attributes by constructing attribute-based hyperedges with appropriate weights, alongside the original hyperedges from the input attributed hypergraph. Importantly, on $\mathcal{H}$, we propose two measures: hypergraph multi-hop node similarity (HMS-N) and hypergraph multi-hop hyperedge similarity (HMS-E). HMS-N captures higher-order connections and global topology between nodes by considering both original and attribute-based hyperedges in $\mathcal{H}$. HMS-E quantifies hyperedge similarities, but on a dual hypergraph of $\mathcal{H}$, where hyperedges are treated as nodes to preserve their multi-hop connections and global features. We then formulate the AHNEE task as an optimization problem with the objective to approximate all-node-pair HMS-N and all-hyperedge-pair HMS-E matrices simultaneously. Directly achieving this objective can be effective but computationally expensive, with time quadratic in the number of nodes and hyperedges. To boost efficiency, `SAHE` unifies the approximations of HMS-N and HMS-E matrices by identifying their shared core computations via theoretical analysis. Despite this unification, the process remains costly for calculation and materialization. To further reduce computational overhead, we develop several optimization techniques that eliminate the need to iteratively materialize large dense matrices, enabling efficient approximation of high-quality node and hyperedge embeddings with guarantees. We conduct extensive experiments on 8 real datasets, comparing `SAHE` against 11 competitors over 3 tasks. The results show that `SAHE` efficiently generates high-utility node and hyperedge embeddings, achieving superior predictive performance in node classification, hyperedge link prediction, and hyperedge classification tasks, while being up to orders of magnitude faster.

In summary, we make the following contributions in the paper.

- We build an attribute-extended hypergraph to incorporate attribute information into hypergraph structures seamlessly, with a careful design to balance both aspects.

- We design two similarity measures HMS-N and HMS-E capturing higher-order connections and global topology of node and hyperedge pairs, respectively. The AHNEE objective is formulated to preserve all-pair HMS-N and HMS-E matrices.

- We develop several techniques to efficiently optimize the objective, including unifying the shared computations of node and hyperedge embeddings, accurate approximation of the similarity matrices, and avoiding iterative dense matrix materialization.

- Extensive experiments on 8 real datasets and 3 downstream tasks demonstrate the effectiveness and efficiency of our method.

Table 4.1: Frequently used notations.

$H = \{\mathcal{V}, \mathcal{E}, \mathbf{X}\}$	An attributed hypergraph H with node set $\mathcal{V}$, hyperedge set $\mathcal{E}$, and node attribute matrix $\mathbf{X} \in \mathbb{R}^{n \times q}$.
n, m	The cardinality of $\lvert\mathcal{V}\rvert = n$, and the cardinality of $\lvert\mathcal{E}\rvert = m$.
$d(v)$	The generalized degree of a node v.
$\delta(e)$	The generalized degree of a hyperedge e.
$\mathcal{H} = \{\mathcal{V}, \mathcal{E}_X\}$	The extended hypergraph H with hyperedge set $\mathcal{E}_X$ that incorporates $\mathcal{E}$ and attribute-based hyperedges $\mathcal{E}_K$.
$\mathrm{vol}(\mathcal{H})$	The volume of the hypergraph $\mathcal{H}$.
$\mathbf{H}$	The weighted incidence matrix of $\mathcal{H}$.
$\mathbf{D}_v$, $\mathbf{D}_e$, $\mathbf{W}$	The diagonal node degree matrix, hyperedge degree matrix, and hyperedge weight matrix of $\mathcal{H}$, respectively.
$\mathcal{H}' = \{\mathcal{V}'_X, \mathcal{E}'\}$	The dual hypergraph of $\mathcal{H}$, where nodes in $\mathcal{V}'_X$ represent hyperedges in $\mathcal{E}_X$, and hyperedges in $\mathcal{E}'$ represent nodes in $\mathcal{V}$.
$p(v_i, v_j)$	The random walk transition probability from v_i to v_j.
$p_s(v)$	The random walk stationary probability of node v.
$\mathbf{P}$, $\mathbf{P}'$	The transition probability matrices of hypergraphs $\mathcal{H}$ and $\mathcal{H}'$.
$\pi(v_i, v_j)$	The probability from v_i to v_j over infinite steps.
$\mathbf{\Pi}^{(t)}$,$\mathbf{\Pi}'^{(t)}$	The t-step RWR matrix for $\mathcal{H}$ and $\mathcal{H}'$, respectively.
$\mathrm{tlog}(\cdot), \mathrm{tlog}^{\circ}(\cdot)$.	$\mathrm{tlog}(x) = \log(\max\{x, 1\})$, $\mathrm{tlog}^{\circ}(\cdot)$ is element-wise $\mathrm{tlog}(\cdot)$.
$\psi(v_i, v_j)$	HMS-N similarity between nodes v_i and v_j of $\mathcal{H}$.
$\psi'(e_i, e_j)$	HMS-E similarity between hyperedges e_i and e_j of $\mathcal{H}$.
$\mathbf{\Psi}$, $\mathbf{\Psi}'$	Similarity matrices for HMS-N and HMS-E, respectively.
$\mathbf{Z}_{\mathcal{V}}$, $\mathbf{Z}_{\mathcal{E}}$	The $n \times k$ node embedding matrix and $m \times k$ hyperedge embedding matrix, respectively.

4.2 Preliminaries

An attributed hypergraph is denoted as $H = \{\mathcal{V}, \mathcal{E}, \mathbf{X}\}$, where $\mathcal{V}$ is a set of n nodes, $\mathcal{E}$ is a set of m hyperedges, and $\mathbf{X} \in \mathbb{R}^{n \times q}$ is the node attribute matrix. Each node v in $\mathcal{V}$ has a q-dimensional attribute vector given by the i-th row of $\mathbf{X}$. A hyperedge $e \in \mathcal{E}$ is a subset of $\mathcal{V}$ containing at least two nodes, and a node v is incident to e if $v \in e$. Figure 4.1 gives an example of attributed hypergraph H with six nodes and three hyperedges (e.g., $e_2 = \{v_3, v_4, v_5\}$), where each node is associated with an attribute vector. Let $\gamma(v, e)$ be the hyperedge-dependent weight of node v in hyperedge e, defaulting to 1 if $v \in e$ and unweighted, or 0 if $v \notin e$. Each hyperedge $e \in \mathcal{E}$ carries a weight $w(e)$, defaulting to 1. The generalized degree of a node $v \in \mathcal{V}$ is $d(v) = \sum_{e \in \mathcal{E}} w(e)\gamma(v, e)$, summing the weighted contributions of v across incident hyperedges. The generalized degree of a hyperedge $e \in \mathcal{E}$ is $\delta(e) = \sum_{v \in e} \gamma(v, e)$, aggregating the weight of nodes within e. The volume of the hypergraph, denoted as $\mathrm{vol}(H) = \sum_{v \in \mathcal{V}} d(v)$, measures its total weighted connectivity by summing the generalized degrees. Let $\mathbf{H}_0 \in \mathbb{R}^{m \times n}$ be the incidence matrix of H, where each entry $\mathbf{H}_0[i, j] = \gamma(v_j, e_i)$ if $v_j \in e_i$, otherwise $\mathbf{H}_0[i, j] = 0$.

Node and Hyperedge Embeddings. For the input H, AHNEE aims to compute an $n \times k$ embedding matrix $\mathbf{Z}_{\mathcal{V}}$ where each row $\mathbf{Z}_{\mathcal{V}}[i]$ is the embedding vector for node $v_i \in \mathcal{V}$ (*node embedding*), and also an $m \times k$ embedding matrix $\mathbf{Z}_{\mathcal{E}}$ where each row $\mathbf{Z}_{\mathcal{E}}[j]$ is the embedding vector for hyperedge $e_j \in \mathcal{E}$ (*hyperedge embedding*).

Tasks. In the attributed hypergraph H, we focus on three significant tasks: performing *node classification* and *hyperedge link prediction* with node embeddings; performing *hyperedge classificaiton* with hyperedge embeddings.

- *Node Classification.* For node v_i, the goal is to predict its class label by feeding its node embedding $\mathbf{Z}_{\mathcal{V}}[i]$ into a trained classifier.

- *Hyperedge Link Prediction.* Given nodes $\{v_i, v_j, ..., v_k\} \subset \mathcal{V}$, the task is to use their node embeddings $\{\mathbf{Z}_{\mathcal{V}}[i], \mathbf{Z}_{\mathcal{V}}[j], ..., \mathbf{Z}_{\mathcal{V}}[k]\}$ to predict whether these nodes form a hyperedge or not.

- *Hyperedge Classification.* For a hyperedge e_i, the goal is to use the hyperedge embedding $\mathbf{Z}_{\mathcal{E}}[i]$ to predict its class label.

Table 4.1 lists the frequently used notations in our paper.

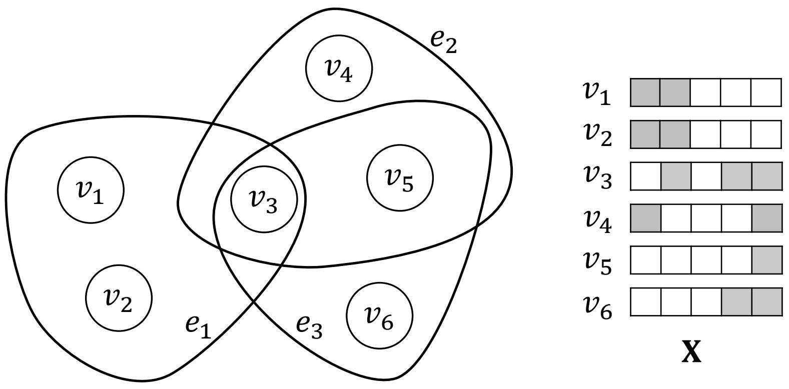
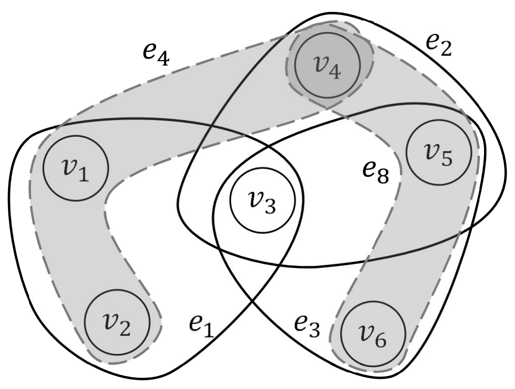


Figure 4.1: An example of attributed hypergraph H.

4.3 Similarities and Objectives

As explained in Section 4.1, effective node and hyperedge embeddings in AHNEE should capture the closeness among nodes and collective affinities among hyperedges, highlighting higher-order structures and attribute influences. To this end, we first extend the attributed hypergraph H by adding attribute-based hyperedges, forming an attribute-extended hypergraph $\mathcal{H}$, in which, appropriate hyperedge-dependent node weights and hyperedge weights are assigned to balance structural and attribute information in Section 4.3.1. Then in Section 4.3.2, we introduce hypergraph multi-hop node similarity (HMS-N) on $\mathcal{H}$ to quantify node similarity. This considers multi-hop node and hyperedge connections and node significance in $\mathcal{H}$, measured by a random walk model over its topology. Section 4.3.3 designs hypergraph multi-hop hyperedge similarity (HMS-E), formulated similarly but on the dual hypergraph of $\mathcal{H}$. Both HMS-N and HMS-E are symmetric for evaluating pairwise relationships. Our AHNEE objective is to approximate all $n \times n$ node-pair HMS-N and all $m \times m$ hyperedge-pair HMS-E similarities. Section 4.3.4 presents a preliminary method to directly solve this problem.



$\mathcal{E}_K = \{e_4, e_5, e_6, e_7, e_8\}$.

$e_4 = \text{KNN}(v_1) \cup \{v_1\}$,
$w(e_4) = 0.38, \gamma(v_1, e_4) = 1.0$,
$\gamma(v_2, e_4) = 1.0, \gamma(v_4, e_4) = 0.5$.

$e_8 = \text{KNN}(v_5) \cup \{v_5\}$,
$w(e_8) = 0.38, \gamma(v_5, e_8) = 1.0$,
$\gamma(v_4, e_8) = 0.7, \gamma(v_6, e_8) = 0.7$.

Figure 4.2: Extended hypergraph $\mathcal{H}$.

4.3.1 Attribute-Extended Hypergraph

The literature on attributed data [73, 29] shows that it is effective for downstream task performance to combine attribute information with network topology, by considering each node's K-nearest neighbors defined on attribute similarity. We adopt this approach to construct an extended hypergraph $\mathcal{H}$ from the input H, with dedicated designs tailored for hypergraph structures and balancing topological and attribute information. Specifically, for a node v_i, we first get its local neighbor set $\text{KNN}(v_i)$, comprising the top-K most similar nodes v_j ranked by cosine similarity $\text{cosSim}(v_i, v_j)$ of their attributes. Then we define an *attribute-based hyperedge* as $\text{KNN}(v_i) \cup \{v_i\}$ with $K + 1$ nodes. Intuitively, it connects nodes with similar attributes into a hyperedge.

Example. Given the hypergraph H in Figure 4.1, we construct five attribute-based hyperedges $\{e_4, \ldots, e_8\}$ for nodes $v_1, \ldots, v_5$, respectively. With $K = 2$, Figure 4.2 illustrates two examples: e_4, formed by v_1 and its two most similar nodes, and e_8, formed by v_5 and its two most similar nodes. Each hyperedge represents a local neighborhood of nodes with high attribute similarity.

For all n nodes in H, we create a set $\mathcal{E}_K$ of n attribute-based hyperedges, leading to the extended hypergraph $\mathcal{H} = \{\mathcal{V}, \mathcal{E}_X\}$ with hyperedge set $\mathcal{E}_X = \mathcal{E} \cup \mathcal{E}_K$ of size $m + n$. Unlike nodes in an original hyperedge, the nodes in an attribute-based hyperedge e

built from $\text{KNN}(v_i)$ require varying weights based on their attribute similarities to v_i. Therefore, we assign *hyperedge-dependent node weights* $\gamma(v_j, e)$ to node v_j in the hyperedge e, using its attribute similarity to v_i: $\gamma(v_j, e) = \text{cosSim}(v_i, v_j)$, for $v_j \in e$ and $e = \text{KNN}(v_i) \cup \{v_i\}$.

$\mathcal{H}$ includes m hyperedges in $\mathcal{E}$ and n attribute-based hyperedges in $\mathcal{E}_K$, representing structural and attribute information, respectively. The values of n and m can vary significantly across datasets. For example, the Amazon dataset has about $n = 2.27$ million nodes and $m = 4.28$ million hyperedges, while MAG-PM has $n = 2.35$ million nodes and $m = 1.08$ million hyperedges. A large n may cause attribute-based hyperedges to overshadow the original topology, and vice versa. This disparity extends to their volumes $\text{vol}(\mathcal{E}) = \sum_{e \in \mathcal{E}} w(e)\delta(e)$. This can influence similarity measures by skewing transitions to $\mathcal{E}$ or $\mathcal{E}_K$, affecting embedding priorities. Thus, we balance structural and attribute insights in $\mathcal{H}$. We achieve this by adjusting hyperedge weights in $\mathcal{E}_K$, balancing the volumes of $\mathcal{E}$ and $\mathcal{E}_K$. For $\mathcal{E}$, the volume is $\text{vol}\,(\mathcal{E}) = \sum_{e \in \mathcal{E}} |e|$, when $\gamma(v, e)$ and $w(e)$ are with unit weights. For $\mathcal{E}_K$, the volume is $\text{vol}(\mathcal{E}_K) = w(e) \sum_{e \in \mathcal{E}_K} \sum_{v \in e} \gamma(v, e)$, where hyperedge-dependent node weight $\gamma(v, e)$ is assigned ahead. We enforce:

$$\beta \,\text{vol}\,(\mathcal{E}) = \text{vol}\,(\mathcal{E}_K)\,, \tag{4.1}$$

where parameter β controls the balance on structure versus attributes, and the default value is 1. Then we get a uniform weight $w(e)$ of each attribute-based hyperedge $e \in \mathcal{E}_K$ by

$$w(e) = \beta \,\text{vol}(\mathcal{E}) \Big/ \sum_{e \in \mathcal{E}_K} \sum_{v \in e} \gamma(v, e), \forall e \in \mathcal{E}_K. \tag{4.2}$$

The extended hypergraph $\mathcal{H} = \{\mathcal{V}, \mathcal{E}_X\}$ has an $(m+n) \times n$ weighted incidence matrix $\mathbf{H}$, where $\mathbf{H}[i, j] = \gamma(v_j, e_i)$. The first m rows of $\mathbf{H}$ are from the incidence matrix $\mathbf{H}_0$, and the last n rows are from the attribute-based hyperedges, forming a submatrix $\mathbf{H}_K$. Then, the matrix $\mathbf{H}$ can be written as $\mathbf{H} = \left[\begin{smallmatrix} \mathbf{H}_0 \\ \mathbf{H}_K \end{smallmatrix}\right]$, where columns corresponding to the nodes in $\mathcal{V}$. For $\mathcal{H}$, let $\mathbf{W} \in \mathbb{R}^{(m+n)\times(m+n)}$ be the diagonal matrix of hyperedge weights. In addition, $\mathbf{D}_v \in \mathbb{R}^{n\times n}$ is the node degree matrix, with $\mathbf{D}_v[i,i] = d(v_i)$ representing the generalized degree of node v_i in $\mathcal{H}$, and $\mathbf{D}_e \in \mathbb{R}^{(m+n)\times(m+n)}$ is the hyperedge degree matrix with $\mathbf{D}_e[i,i] = \delta(e_i)$ representing the generalized degree of hyperedge e_i in $\mathcal{H}$.

The above idea of constructing $\mathcal{H}$ from $H = \{\mathcal{V}, \mathcal{E}, \mathbf{X}\}$ is summarized in Algorithm 7. Lines 1-4 generate n attribute-based hyperedges $e_i = \mathrm{KNN}(v_i) \cup \{v_i\}$, each capturing a node's K-nearest neighbors based on attribute similarity from $\mathbf{X}$ (Line 2). Lines 3-4 assign hyperedge-dependent node weights $\gamma(v_j, e_i)$ in $\mathbf{H}_K$. Line 5 constructs the hyperedge weight matrix $\mathbf{W}$ as a diagonal matrix from concatenated weight vectors (denoted by $\|$). The m hyperedges in $\mathcal{E}$ have weights of 1 via $\mathbf{1}_m$, a length-m vector of ones, while the n attribute-based hyperedges receive a weight per Eq. (4.2), with $\mathbf{1}_m\mathbf{H}_0\mathbf{1}_n$ representing $\mathrm{vol}(\mathcal{E})$. Line 6 builds the $(m+n)\times n$ weighted incidence matrix $\mathbf{H}$ by stacking $\mathbf{H}_0$ and $\mathbf{H}_K$, and computes the diagonal degree matrices $\mathbf{D}_v$ and $\mathbf{D}_e$ of $\mathcal{H}$. The algorithm returns $\mathbf{H}$, $\mathbf{D}_v$, $\mathbf{D}_e$, and $\mathbf{W}$, representing $\mathcal{H}$. Lines 1-4 require $O(n\log n + nqK)$ time, leveraging efficient KNN queries (*e.g.*, [21]), where q is the attribute dimension. Lines 5-6 operate in $O(nK + n\bar{d})$ time and space, proportional to $\mathbf{H}$'s nonzero entries, with $\bar{d}$ as the average hyperedge incidences per node in H. Thus, Algorithm 7 achieves log-linear time complexity and linear space complexity for generating $\mathcal{H}$.

4.3.2 Hypergraph Multi-Hop Node Similarity: HMS-N

Intuitively, the resultant node embeddings should capture the complex relationships between nodes, preserving both structural and attribute similarities across the extended hypergraph $\mathcal{H}$. This is challenging due to the need to model higher-order connections in hyperedges while integrating the hypergraph's global topology. Common similarity measures like Personalized PageRank [140] effectively capture node

```
Algorithm 7: ExtendHG
Input: Attributed hypergraph $H = \{\mathcal{V}, \mathcal{E}, \mathbf{X}\}$, parameter $K$
1 for each $v_i \in \mathcal{E}$ do
2     Attribute-based hyperedge $e_i \leftarrow \mathrm{KNN}(v_i) \cup \{v_i\}$;
3     for each $v_j \in e_i$ do
4         $\mathbf{H}_K[i,j] \leftarrow \gamma(v_j, e_i)$;
5 $\mathbf{W} \leftarrow \mathrm{diag}\left(\mathbf{1}_m \| \frac{\mathbf{1}_m^\top \mathbf{H}_0 \mathbf{1}_n}{\mathbf{1}_n^\top \mathbf{H}_K \mathbf{1}_n} \mathbf{1}_n\right)$;
6 $\mathbf{H} \leftarrow \left[\begin{smallmatrix}\mathbf{H}_0 \\ \mathbf{H}_K\end{smallmatrix}\right]$, $\mathbf{D}_v \leftarrow \mathrm{diag}\left(\mathbf{H}^\top \mathbf{W} \mathbf{1}_{m+n}\right)$, $\mathbf{D}_e \leftarrow \mathrm{diag}\left(\mathbf{H}\mathbf{1}_n\right)$;
7 return Extended hypergraph $\mathcal{H}$ (i.e., $\mathbf{H}$, $\mathbf{D}_v$, $\mathbf{D}_e$, $\mathbf{W}$);
```

significance but are limited to pairwise interactions, not applicable to hypergraphs. Other hypergraph definitions [2, 113] consider submodular hyperedges, which are unnecessary for embeddings and incur expensive all-pair computations.

To address these challenges, we propose a hypergraph multi-hop similarity measure for nodes (HMS-N) over $\mathcal{H}$. The key insights include (i) capturing multi-hop connectivity between nodes and (ii) leveraging the global significance of nodes in $\mathcal{H}$, both relying on random walks adapted to the hypergraph structure.

HMS-N Formulation. We begin by defining the transition probability $p(u, v)$ for random walks on $\mathcal{H}$, considering hyperedge sizes, weights $w(e)$, generalized node degrees $d(u)$, and, crucially, hyperedge-dependent node weights $\gamma(u, e)$, reflecting attribute similarities. $p(u, v)$ involves two hops: from node u to a hyperedge and from the hyperedge to node v. First, unlike prior definitions [17], an incident hyperedge e is selected with probability proportional to $w(e)\gamma(u, e)/d(u)$, where $\gamma(u, e)$ emphasizes attribute affinity, and $w(e)$ balances structural and attribute significance. Second, within the chosen hyperedge e, the node v is selected with probability proportional to $\gamma(v, e)/\delta(e)$, prioritizing nodes with stronger attribute ties. Therefore, the transition probability is

$$p(u, v) = \sum_{e \in \mathcal{E}_X} \frac{w(e)\gamma(u, e)}{d(u)} \frac{\gamma(v, e)}{\delta(e)}, \tag{4.3}$$

and accordingly the $n \times n$ transition matrix $\mathbf{P}$ with each entry $\mathbf{P}[i,j] = p(v_i, v_j)$ can be written as

$$\mathbf{P} = \mathbf{D}_v^{-1}\mathbf{H}^{\mathsf{T}}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{H}. \tag{4.4}$$

The HMS-N measure $\psi(\cdot)$ combines the multi-hop connectivity derived from random walks, and the global significance of nodes featured by the stationary probability.

First, to capture the connectivity between nodes, we carry out a number of random walks, each of which may restart from its beginning node to balance the local and global structural insights. Specifically, in the extended hypergraph $\mathcal{H}$, at each step, the walk either teleports back to u with probability $\alpha \in [0, 1)$ or transitions to a node with probability $1 - \alpha$, following the transition matrix $\mathbf{P}$ in Equation (4.4). Let $\pi(v_i, v_j)$ denote the limiting probability that a random walk starting from node v_i reaches node v_j after infinitely many iterations, reflecting v_j's significance to v_i across local and global levels. The probability $\pi^{(t)}(v_i, v_j)$ of reaching any node v_j from any node v_i after t steps is represented by the stochastic matrix $\mathbf{\Pi}^{(t)} \in \mathbb{R}^{n \times n}$ in Equation (4.5), where $\mathbf{\Pi}^{(t)}[i,j]$ is $\pi^{(t)}(v_i, v_j)$.

$$\mathbf{\Pi}^{(0)} = \mathbf{I}_n, \;\; \mathbf{\Pi}^{(t+1)} = \alpha\mathbf{I}_n + (1-\alpha)\mathbf{P}\mathbf{\Pi}^{(t)}, \tag{4.5}$$

with $\mathbf{I}_n$ as the $n \times n$ identity matrix. The non-recursive formula is

$$\mathbf{\Pi}^{(t)} = \sum_{i=0}^{t-1} \alpha(1-\alpha)^i\mathbf{P}^i + (1-\alpha)^t\mathbf{P}^t. \tag{4.6}$$

Let $\mathbf{\Pi}$ represent $\mathbf{\Pi}^{(t=\infty)}$ with infinite steps to converge. As one may note, Equations (4.5) and (4.6) have similar forms to those in simple graphs. However, our formulation is based on the dedicatedly derived from the extended hypergraph $\mathcal{H}$. Accordingly, the transition includes the selection of a hyperedge and a node therein. Moreover, we do not employ $\mathbf{\Pi}$ as the similarity matrix, but instead take account of the significance of each specific node as below.

Second, note that in a connected and nontrivial $\mathcal{H}$, the transition matrix $\mathbf{P}$ is irreducible and aperiodic, ensuring a unique stationary distribution $\mathbf{p}_s = \mathbf{p}_s\mathbf{P}$ [156]. Let $p_s(v)$ be the element in $\mathbf{p}_s$ w.r.t. node v. Then, $p_s(v)$ is the probability that an arbitrary random walk ends at v, indicating the significance of v. Moreover, the significance $p_s(v)$ can be calculated by $p_s(v) = d(v)/\operatorname{vol}(\mathcal{H})$.

Finally, combining the multi-hop connectivity and node significance, we define the HMS-N similarity measure as

$$\psi(u,v) = \operatorname{tlog}\frac{\pi(u,v)}{p_s(v)}, \tag{4.7}$$

where $\pi(u,v)$ is divided by $p_s(v)$ to offset the inherent global significance of v (*i.e.*, its degree centrality), and thus isolate the specific relational strength between u and v for a balanced and embedding-friendly measure. Moreover, the truncated logarithm, $\operatorname{tlog} x = \log(\max(x,1))$, is applied to stabilize the ratio against small $p_s(v)$ in large hypergraphs.

By Lemma 4.3.1 proved below, we establish the symmetry of HMS-N, ensuring the mutual similarity between two nodes. This is critical for embedding, as the dot product of corresponding embedding vectors should preserve their mutual HMS-N similarity.

Lemma 4.3.1. *For any nodes $v_i, v_j \in \mathcal{V}$, we have $\frac{\pi(v_i,v_j)}{p_s(v_j)} = \frac{\pi(v_j,v_i)}{p_s(v_i)}$.*

Proof. The random walk on $\mathcal{H}$ has stationary probability $p_s(v_i) = d(v_i)/\operatorname{vol}(\mathcal{H})$ [156]. Thus, the matrix with $\mathbf{p}_s$ as diagonal is $\mathbf{D}_v/\operatorname{vol}(\mathcal{H})$. Two sides of the lemma are written in matrix form as $\operatorname{vol}(\mathcal{H})\mathbf{\Pi}\mathbf{D}_v^{-1}$ and $(\operatorname{vol}(\mathcal{H})\mathbf{\Pi}\mathbf{D}_v^{-1})^\top$. Then we get $\mathbf{D}_v^{-1}\mathbf{\Pi}^\top = \mathbf{D}_v^{-1}\sum_{i=0}^{\infty}\alpha(1-\alpha)^i(\mathbf{P}^i)^\top = \sum_{i=0}^{\infty}\alpha(1-\alpha)^i\mathbf{D}_v^{-1}\left(\mathbf{H}^\top\mathbf{W}\mathbf{D}_e^{-1}\mathbf{H}\mathbf{D}_v^{-1}\right)^i = \sum_{i=0}^{\infty}\alpha(1-\alpha)^i\mathbf{P}^i\mathbf{D}_v^{-1} = \mathbf{\Pi}\mathbf{D}_v^{-1} = (\mathbf{D}_v^{-1}\mathbf{\Pi}^\top)^\top$. □

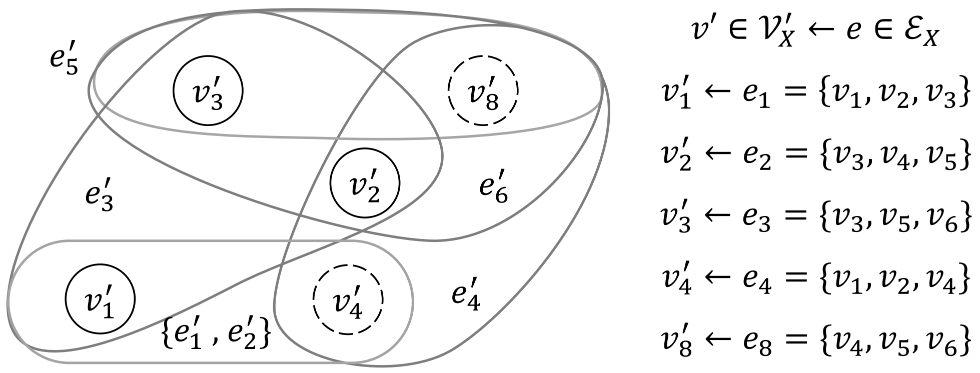

e'_5
v'_3
v'_8
e'_3
v'_2
e'_6
v'_1
v'_4
e'_4
$\{e'_1, e'_2\}$
$v' \in \mathcal{V}'_X \leftarrow e \in \mathcal{E}_X$
$v'_1 \leftarrow e_1 = \{v_1, v_2, v_3\}$
$v'_2 \leftarrow e_2 = \{v_3, v_4, v_5\}$
$v'_3 \leftarrow e_3 = \{v_3, v_5, v_6\}$
$v'_4 \leftarrow e_4 = \{v_1, v_2, v_4\}$
$v'_8 \leftarrow e_8 = \{v_4, v_5, v_6\}$

Figure 4.3: Dual hypergraph $\mathcal{H}'$.

With Eq. (4.7), we can express the HMS-N similarity matrix for all $n \times n$ node pairs in $\mathcal{H}$ as

$$\mathbf{\Psi} = \text{tlog}^{\circ}\left(\text{vol}(\mathcal{H})\mathbf{\Pi}\mathbf{D}_v^{-1}\right), \tag{4.8}$$

where $\mathbf{\Psi}[i,j] = \psi(v_i, v_j)$ and $\text{tlog}^{\circ}(\cdot)$ means the element-wise truncated logarithm. Also, $\mathbf{\Psi}$ is a symmetric matrix, as the element-wise $\text{tlog}^{\circ}(\cdot)$ function preserves the symmetry established in Lemma 4.3.1.

Node Embedding Objective. We aim to use the dot product of two node embeddings to preserve the HMS-N between nodes. Specifically, Let $\mathbf{Z}_{\mathcal{V}}$ denote the $n \times k$ embedding matrix where each row is a node embedding vector. Then, the node embedding problem of solving $\mathbf{Z}_{\mathcal{V}}$ can be formulated as follows, where $\|\cdot\|_F$ is the Frobenius norm.

$$\mathbf{Z}_{\mathcal{V}} = \text{argmin}_{\mathbf{Z} \in \mathbb{R}^{n \times k}} \|\mathbf{\Psi} - \mathbf{Z}\mathbf{Z}^{\intercal}\|_F^2. \tag{4.9}$$

4.3.3 Hypergraph Multi-Hop Hyperedge Similarity: HMS-E

The definition of HMS-E aligns with the principles of HMS-N, but it is derived using the dual hypergraph $\mathcal{H}'$ of $\mathcal{H}$, where nodes and hyperedges swap their roles. $\mathcal{H}'$ can be obtained by transposing the incidence matrix $\mathbf{H}$. In the dual hypergraph $\mathcal{H}' = \{\mathcal{V}'_X, \mathcal{E}'\}$, the new node set $\mathcal{V}'_X$ includes $m + n$ nodes and the new hyperedge set

$\mathcal{E}'$ contains n hyperedges. Specifically, each node $v'_i \in \mathcal{V}'_X$ represents the hyperedge $e_i \in \mathcal{E}_X$ of $\mathcal{H}$. Each hyperedge $e'_j \in \mathcal{E}'$ contains nodes in $\mathcal{V}'_X$ which correspond to the hyperedges in $\mathcal{H}$ incident to $v_j \in \mathcal{V}$.

Example. Figure 4.3 illustrates the dual hypergraph $\mathcal{H}'$ derived from the original $\mathcal{H}$ in Figure 4.2. In $\mathcal{H}'$, hyperedge e_1 of $\mathcal{H}$ is represented by node v'_1 and is connected to v'_2 and v'_3 through e'_3, which corresponds to node v_3 in $\mathcal{H}$. This representation enables the analysis of similarity between e_1 and e_2 in $\mathcal{H}$ based on the relationships between v'_1 and v'_2 in $\mathcal{H}'$.

The dual hypergraph $\mathcal{H}'$ has a weighted incidence matrix $\mathbf{H}' = \mathbf{H}^\mathsf{T}\mathbf{W}$, hyperedge weight $w(e') = 1$ ($e' \in \mathcal{E}'$), and $\mathbf{W}' = \mathbf{I}_n$. Hyperedge-dependent node weights are $\gamma'(v'_i, e'_j) = w(e_i)\gamma(v_j, e_i)$, retaining the influence of hyperedge weights in $\mathcal{H}$. The generalized node degree, hyperedge degree, and hypergraph volume of $\mathcal{H}'$ are:

$$d(v'_i) = \delta(e_i)w(e_i), \delta(e'_j) = d(v_j), \mathrm{vol}(\mathcal{H}') = \mathrm{vol}(\mathcal{H}),$$

with matrix forms $\mathbf{D}'_v = \mathbf{W}\mathbf{D}_e$ and $\mathbf{D}'_e = \mathbf{D}_v$.

HMS-E Formulation. In the dual hypergraph $\mathcal{H}'$, a random walk transitions from node v'_i (hyperedge e_i in $\mathcal{H}$) to node v'_j (hyperedge e_j in $\mathcal{H}$) via a hyperedge e' (node v shared by e_i and e_j), with probability proportional to $\gamma'(v'_i, e')\gamma'(v'_j, e')/d(v'_i)$, which is $\gamma(v, e_i)\gamma(v, e_j)/[\delta(e_i)w(e_i)]$ in $\mathcal{H}$. Aggregating over all shared nodes, the transition probability between e_i and e_j effectively captures the strength of their overlap. The transition probability between hyperedges e_i, e_j in the original hypergraph $\mathcal{H}$ is

$$p'(e_i, e_j) = p(v'_i, v'_j) = \sum_{v \in \mathcal{V}} \frac{\gamma(v, e_i)}{\delta(e_i)w(e_i)} \frac{\gamma(v, e_j)}{d(v)}, \tag{4.10}$$

where the function $p'(\cdot, \cdot)$ with a prime indicates the transition between hyperedges in the original hypergraph $\mathcal{H}$. Moreover, the $(m+n) \times (m+n)$ transition matrix $\mathbf{P}'$

can be expressed as

$$\mathbf{P}' = (\mathbf{D}_e\mathbf{W})^{-1}(\mathbf{H}^\intercal\mathbf{W})^\intercal\mathbf{D}_v^{-1}\mathbf{H}^\intercal\mathbf{W} = \mathbf{D}_e^{-1}\mathbf{H}\mathbf{D}_v^{-1}\mathbf{H}^\intercal\mathbf{W}. \tag{4.11}$$

Similar to HMS-N, HMS-E between hyperedges e_i and e_j (i.e., nodes v_i' and v_j' in $\mathcal{H}'$) also considers their multi-hop connectivity and global significance within $\mathcal{H}'$. Let $\mathbf{\Pi}'^{(t)}$ be the probability of transitioning between hyperedges (represented as nodes in $\mathcal{H}'$) over t steps. In the limit, $\mathbf{\Pi}'^{(\infty)}$ reflects the long-term likelihood of reaching one hyperedge from another. The matrix $\mathbf{\Pi}'^{(t)}$ for the dual hypergraph is computed iteratively by substituting $\mathbf{P}'$ into Eq. (4.5). Let $\mathbf{\Pi}'$ denote $\mathbf{\Pi}'^{(t=\infty)}$ with infinite steps to converge. Then, the probability that a random walk from hyperedge e_i reaching hyperedge e_j in $\mathcal{H}$ is $\pi'(e_i, e_j) = \pi(v_i', v_j') = \mathbf{\Pi}'[i, j]$.

The global significance of hyperedge e_j is the significance of its corresponding node v_j' in $\mathcal{H}'$, $p_s(v_i')$, calculated as :

$$p_s'(e_i) = p_s(v_i') = d(v_i')/\operatorname{vol}(\mathcal{H}') = \delta(e_i)w(e_i)/\operatorname{vol}(\mathcal{H}). \tag{4.12}$$

Finally, the HMS-E of hyperedges e_i and e_j, $\psi'(e_i, e_j)$, is

$$\psi'(e_i, e_j) = \operatorname{tlog}\frac{\pi'(e_i, e_j)}{p_s'(e_j)}. \tag{4.13}$$

The corresponding HMS-E matrix for all hyperedge pairs is

$$\mathbf{\Psi}' = \operatorname{tlog}^\circ\left(\operatorname{vol}(\mathcal{H})\mathbf{\Pi}'\mathbf{D}_e^{-1}\mathbf{W}^{-1}\right). \tag{4.14}$$

Hyperedge Embedding Objective. We aim to use the dot product of two hyperedge embeddings to preserve the HMS-E between the hyperedges. Let $\mathbf{Z}_\mathcal{E}$ denote the $m \times k$ embedding matrix where each row is the embedding vector for a hyperedge $e \in \mathcal{E}$, and $\mathbf{\Psi}_\mathcal{E}'$ denote the $m \times m$ submatrix of $\mathbf{\Psi}'$ that represents the HMS-E similarity between hyperedges in $\mathcal{E}$. Then, the hyperedge embedding problem of solving $\mathbf{Z}_{\mathcal{E}}$ can be formulated as

$$\mathbf{Z}_{\mathcal{E}} = \operatorname{argmin}_{\mathbf{Z}\in\mathbb{R}^{m\times k}} \|\mathbf{\Psi}'_{\mathcal{E}} - \mathbf{Z}\mathbf{Z}^{\mathsf{T}}\|_F^2. \tag{4.15}$$

4.3.4 A Base Method

In this section, we introduce a basic method to directly address the node and hyperedge embedding objectives in Equations (4.9) and (4.15). The purpose of this base method is two-fold. First, it verifies the effectiveness of the proposed measures HMS-N and HMS-E, by demonstrating superior quality over existing methods. Second, it establishes a foundation for our final method `SAHE`, described in Section 4.4, which achieves the same high embedding quality with significantly improved efficiency.

For node embeddings, the main idea is factorizing the HMS-N matrix $\mathbf{\Psi}$. Recall that $\mathbf{\Psi}$ in Eq. (4.8) relies on $\mathbf{\Pi}^{(t=\infty)}$ in Eq. (4.5), which is an infinite sum of powers of the transition matrix $\mathbf{P}$. To be tractable, we approximate $\mathbf{\Pi}$ by at most $t = T$ steps, resulting in $\mathbf{\Pi}^{(T)}$ by Eq. (4.6). Accordingly, we can derive the approximate HMS-N matrix $\mathbf{\Psi}_T$ by replacing $\mathbf{\Pi}$ with $\mathbf{\Pi}^{(T)}$ in Eq. (4.8). Now, the focus is to factorize the symmetric matrix $\mathbf{\Psi}_T$ to get node embeddings $\mathbf{Z}_{\mathcal{V}} \in \mathbb{R}^{n\times k}$. Specifically, we utilize the eigendecomposition $\mathbf{\Psi}_T = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{\mathsf{T}}$, where $\mathbf{\Lambda}$ is the diagonal matrix of n eigenvalues, and $\mathbf{Q}$ contains the corresponding eigenvectors in its columns. Then, the embedding matrix $\mathbf{Z}_{\mathcal{V}}$ would be $\mathbf{Q}\mathbf{\Lambda}^{1/2}$, so that $\mathbf{Z}_{\mathcal{V}}\mathbf{Z}_{\mathcal{V}}^{\mathsf{T}}$ approximates $\mathbf{\Psi}$ in Eq. (4.9). Note that, $\mathbf{Z}_{\mathcal{V}}$ has only k columns. To satisfy this, we only take the k leading eigenvalues (forming $\mathbf{\Lambda}_{\Psi}$), and let $\mathbf{Q}_k$ contain the corresponding eigenvectors. Then, we get the node embeddings

$$\mathbf{Z}_{\mathcal{V}} = \mathbf{Q}_k\mathbf{\Lambda}_{\Psi}^{1/2}, \text{ where } \mathbf{Q}_k \in \mathbb{R}^{n\times k},\ \mathbf{\Lambda}_{\Psi} \in \mathbb{R}^{k\times k}. \tag{4.16}$$

Algorithm 8: `Base`

Input: Hypergraph incidence matrix $\mathbf{H}_0 \in \mathbb{R}^{m\times n}$ and attribute matrix $\mathbf{X} \in \mathbb{R}^{n\times q}$, embedding dimension k, algorithm parameters K, α, T.

```
1  $\mathbf{H}, \mathbf{D}_v, \mathbf{D}_e, \mathbf{W} \leftarrow \texttt{ExtendHG}(\mathbf{H}_0, \mathbf{X}, K)$;
2  $\mathbf{P} \leftarrow \mathbf{D}_v^{-1}\mathbf{H}^\top\mathbf{W}\mathbf{D}_e^{-1}\mathbf{H},\ \mathbf{\Pi}^{(0)} \leftarrow \mathbf{I}_n$ ;          // Eq. (4.4)
3  for $t \leftarrow 1,\dots,T$ do
4      $\mathbf{\Pi}^{(t)} \leftarrow \alpha\mathbf{I}_n + (1-\alpha)\mathbf{P}\mathbf{\Pi}^{(t-1)}$ ;          // Eq. (4.5)
5  $\mathbf{\Psi}_T \leftarrow \mathrm{tlog}^\circ\left(\mathrm{vol}(\mathcal{H})\mathbf{\Pi}^{(T)}\mathbf{D}_v^{-1}\right)$ ;
6  $\mathbf{\Lambda}_\Psi, \mathbf{Q}_k \leftarrow \mathrm{eigen}(\mathbf{\Psi}_T, k)$;
7  $\mathbf{Z}_\mathcal{V} \leftarrow \mathbf{Q}_k\mathbf{\Lambda}_\Psi^{1/2}$ ;          // Eq. (4.16)
8  $\mathbf{P}' \leftarrow \mathbf{D}_e^{-1}\mathbf{H}\mathbf{D}_v^{-1}\mathbf{H}^\top\mathbf{W},\ \mathbf{\Pi}'^{(0)} \leftarrow \mathbf{I}_{m+n}$;
9  for $t \leftarrow 1,\dots,T$ do
10     $\mathbf{\Pi}'^{(t)} \leftarrow \alpha\mathbf{I}_{m+n} + (1-\alpha)\mathbf{P}'\mathbf{\Pi}'^{(t-1)}$;
11 $\mathbf{\Psi}'_T \leftarrow \mathrm{tlog}^\circ\left(\mathrm{vol}(\mathcal{H})\mathbf{\Pi}'^{(T)}\mathbf{D}_e^{-1}\mathbf{W}^{-1}\right)$;
12 $\mathbf{\Psi}'_\mathcal{E} \leftarrow \mathbf{\Psi}'_T[1:m+1, 1:m+1]$;
13 $\mathbf{\Lambda}'_\Psi, \mathbf{Q}'_k \leftarrow \mathrm{eigen}(\mathbf{\Psi}'_\mathcal{E}, k)$;
14 $\mathbf{Z}_\mathcal{E} \leftarrow \mathbf{Q}'_k\mathbf{\Lambda}'^{1/2}_\Psi$;
15 return $\mathbf{Z}_\mathcal{V}, \mathbf{Z}_\mathcal{E}$;
```

Similarly, to derive the hyperedge embeddings $\mathbf{Z}_\mathcal{E}$ in Eq. (4.15), the base method first gets $\mathbf{\Pi}'^{(T)}$ with at most T steps, then computes the approximate HMS-E matrix $\mathbf{\Psi}'_T$. A note is that we are only interested in deriving embeddings for hyperedges in $\mathcal{E}$, while the attribute-based hyperedges in $\mathcal{E}_K$ are constructed just to incorporate the attributes. Thus, we only focus on factorizing the $m\times m$ part of $\mathbf{\Psi}'$ (denoted as $\mathbf{\Psi}'_\mathcal{E}$). Although these attribute-based hyperedges are excluded from $\mathbf{\Psi}'_\mathcal{E}$, the attribute information is actually taken into account in $\mathbf{\Psi}'_\mathcal{E}$ via the multi-hop random walk. Then, after factorizing $\mathbf{\Psi}'_\mathcal{E} = \mathbf{Q}'\mathbf{\Lambda}'\mathbf{Q}'^\top$, we take the first k leading eigenvalues (forming $\mathbf{\Lambda}'_\Psi$) and the corresponding eigenvectors (forming $\mathbf{Q}'_k$) to fit the dimension of embeddings k. Finally, we can derive the hyperedge embeddings $\mathbf{Z}_\mathcal{E} = \mathbf{Q}'_k\mathbf{\Lambda}'^{1/2}_\Psi$.

The pseudocode of this method is presented in Algorithm 8. After constructing the

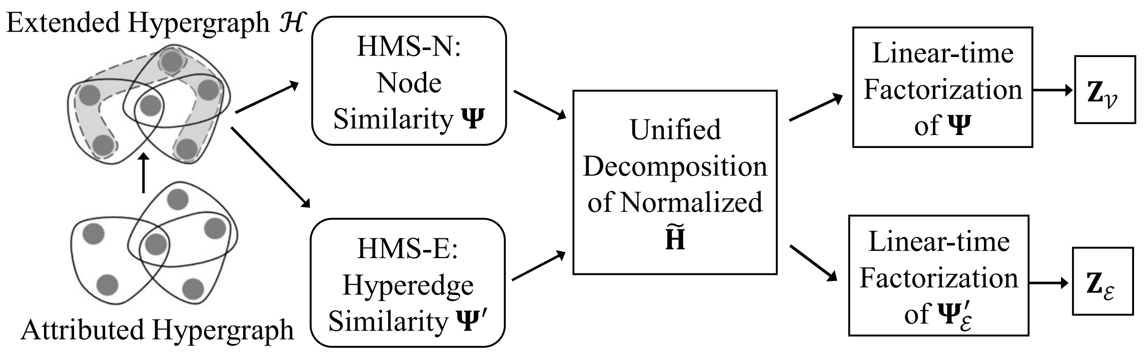

Extended Hypergraph $\mathcal{H}$
Attributed Hypergraph
HMS-N: Node Similarity $\mathbf{\Psi}$
HMS-E: Hyperedge Similarity $\mathbf{\Psi}'$
Unified Decomposition of Normalized $\widetilde{\mathbf{H}}$
Linear-time Factorization of $\mathbf{\Psi}$
Linear-time Factorization of $\mathbf{\Psi}'_{\mathcal{E}}$
$\mathbf{Z}_{\mathcal{V}}$
$\mathbf{Z}_{\mathcal{E}}$

Figure 4.4: Overview of the SAHE algorithm.

extended hypergraph $\mathcal{H}$ by Line 1, Base first simulates the random walk processes on $\mathcal{H}$ for T iterations in Lines 2-4, leading to $\mathbf{\Pi}^{(T)}$. The approximate HMS-N matrix $\mathbf{\Psi}_T$ is computed in Line 5, and the node embedding matrix $\mathbf{Z}_{\mathcal{V}}$ is derived by factorizing $\mathbf{\Psi}_T$ in Lines 6-7. For the eigendecomposition in Line 6, we adopt an implementation based on Lanczos iterations, which solves the leading eigenpairs via a limited number of matrix-vector multiplications. Then, Lines 8-14 basically repeat the embedding procedures on the dual hypergraph $\mathcal{H}'$ to acquire the hyperedge embeddings $\mathbf{Z}_{\mathcal{E}}$, except that Line 12 removes the last n columns and rows to exclude the attribute-based hyperedges from $\mathbf{H}$.

In the experiments, Base demonstrates strong effectiveness over existing methods, but falls short in scalability. To analyze, Lines 3-4 and Lines 9-10 dominate its time complexity, with running time $O(n^2\bar{d}^2)$ and $O((m+n)^2\bar{d}^2)$, respectively. Lines 5-6 and Lines 11-13 also incur quadratic time while handling $\mathbf{\Psi}_T$ and $\mathbf{\Psi}_{\mathcal{E}}$. Thus, Base has an overall time complexity of $O((m+n)^2\bar{d}^2)$, and the space complexity is $O((m+n)^2)$, due to the materialization of the $(m+n)\times(m+n)$ matrix $\mathbf{\Pi}'$ and the $n\times n$ matrix $\mathbf{\Pi}$. The high complexity stems from the element-wise $\operatorname{tlog}^{\circ}(\cdot)$ function, which prevents separate factorization of $\mathbf{\Pi}$ and $\mathbf{D}^{-1}$, forcing materialization of $\mathbf{\Pi}^{(t)}$ for $t \in [1, T]$. Moreover, $\mathbf{\Pi}^{(t)}$ and $\mathbf{\Pi}'^{(t)}$ grow dense after iterations, exacerbating scalability issues. To overcome these limitations, we design SAHE in Section 4.4.

4.4 The SAHE Method

As explained, materializing the dense HMS-N and HMS-E matrices $\mathbf{\Pi}^{(T)}$ and $\mathbf{\Pi}'^{(T)}$ is computationally expensive. The non-linearity in the definitions of HMS-N and HMS-E complicates straightforward matrix factorization of the transition and incidence matrices used to construct the similarity matrices.

To solve these difficulties, we develop SAHE, an efficient method to produce high-quality AHNEE results, with a complete pipeline outlined by Figure 4.4. The key ideas are two-fold. First, we analyze the shared core computations of HMS-N and HMS-E matrices, enabling a unified matrix decomposition procedure for node and hyperedge embedding objectives (Section 4.4.1). Second, we introduce approximation techniques to efficiently generate node and hyperedge embeddings in linear time, avoiding the materialization of dense HMS-N and HMS-E similarity matrices, with theoretical guarantees for the approximations (Section 4.4.2). In Section 4.4.3, we present the algorithmic details.

4.4.1 Unify HMS-N and HMS-E Computations

The key strategy is to identify the shared core computations of the HMS-N and HMS-E matrices, and perform early matrix decomposition to avoid materializing $\mathbf{\Pi}^{(T)}$ and $\mathbf{\Pi}'^{(T)}$, thereby improving efficiency. For node embedding, to derive the HMS-N matrix $\mathbf{\Psi}_T$, according to Eq. (4.6) and Eq. (4.8), we need to obtain $\mathbf{\Pi}^{(T)}\mathbf{D}_v^{-1}$ first, as follows.

$$\mathbf{\Pi}^{(T)}\mathbf{D}_v^{-1} = \sum_{i=0}^{T-1} \alpha(1-\alpha)^i \mathbf{P}^i \mathbf{D}_v^{-1} + (1-\alpha)^T \mathbf{P}^T \mathbf{D}_v^{-1}.$$

The main computation here is to get the term $\mathbf{P}^i\mathbf{D}_v^{-1}$ ($i \in [T]$). We reformulate $\mathbf{P}^i\mathbf{D}_v^{-1}$ by plugging in the definition of $\mathbf{P}$ in Eq. (4.4) as follows, and obviously

$\mathbf{P}^i\mathbf{D}_v^{-1}$ is symmetric.

$$\mathbf{P}^i\mathbf{D}_v^{-1} = \mathbf{D}_v^{-1/2}\left(\widetilde{\mathbf{H}}^\top\widetilde{\mathbf{H}}\right)^i\mathbf{D}_v^{-1/2}, \tag{4.17}$$

where $\widetilde{\mathbf{H}} = \mathbf{W}^{1/2}\mathbf{D}_e^{-1/2}\mathbf{H}\mathbf{D}_v^{-1/2}$ and $\widetilde{\mathbf{H}} \in \mathbb{R}^{(m+n)\times n}$.

For hyperedge embedding, similarly, to derive the HMS-E matrix $\mathbf{\Psi}'_T$, according to the formulation in Section 4.3.3, we need to obtain $\mathbf{\Pi}'^{(T)}\mathbf{D}_e^{-1}$, which relies on a recurring symmetric matrix $\mathbf{P}'^i\mathbf{D}_e^{-1}\mathbf{W}^{-1}$ that can be decomposed as

$$\mathbf{P}'^i\mathbf{D}_e^{-1}\mathbf{W}^{-1} = \mathbf{D}_e^{-1/2}\mathbf{W}^{-1/2}\left(\widetilde{\mathbf{H}}\widetilde{\mathbf{H}}^\top\right)^i\mathbf{D}_e^{-1/2}\mathbf{W}^{-1/2}. \tag{4.18}$$

Importantly, observe that in Eq. (4.17) and Eq. (4.18), they both rely on matrix $\widetilde{\mathbf{H}} \in \mathbb{R}^{(m+n)\times n}$, which is essentially a normalized version of the incidence matrix $\mathbf{H}$. Specifically, both HMS-N and HMS-E matrices rely on $\widetilde{\mathbf{H}}$ to get either $\left(\widetilde{\mathbf{H}}^\top\widetilde{\mathbf{H}}\right)^i$ or $\left(\widetilde{\mathbf{H}}\widetilde{\mathbf{H}}^\top\right)^i$ for i up to T.

Note that $\widetilde{\mathbf{H}}$ is sparse since $\mathbf{H}$ is typically sparse. Therefore, it is fast to decompose $\widetilde{\mathbf{H}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ by reduced singular value decomposition (RSVD), where $\mathbf{\Sigma}$ is an $n \times n$ diagonal matrix containing the first n singular values of $\widetilde{\mathbf{H}}$, while $\mathbf{U} \in \mathbb{R}^{(m+n)\times n}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ contain the associated left and right singular vectors as their rows, respectively. Then, $\mathbf{P}^i\mathbf{D}_v^{-1}$ in Eq. (4.17) is formulated as

$$\begin{aligned}\mathbf{P}^i\mathbf{D}_v^{-1} &= \mathbf{D}_v^{-1/2}\left(\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top\right)^i\mathbf{D}_v^{-1/2}\\ &= \mathbf{D}_v^{-1/2}\left(\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top\right)^i\mathbf{D}_v^{-1/2}\\ &= \mathbf{D}_v^{-1/2}\mathbf{V}\mathbf{\Sigma}^{2i}\mathbf{V}^\top\mathbf{D}_v^{-1/2},\end{aligned}$$

where the second and third equalities hold since the singular vectors are orthonormal (i.e., $\mathbf{U}\mathbf{U}^\top = \mathbf{I}_{m+n}$ and $\mathbf{V}\mathbf{V}^\top = \mathbf{I}_n$).

Accordingly, the HMS-N matrix $\boldsymbol{\Psi}_T$ can be written as

$$\begin{aligned}
\boldsymbol{\Psi}_T &= \operatorname{tlog}^{\circ}\left(\operatorname{vol}(\mathcal{H})\boldsymbol{\Pi}^T\mathbf{D}_v^{-1}\right) \\
&= \operatorname{tlog}^{\circ}\left[\operatorname{vol}(\mathcal{H})\left(\sum_{i=0}^{T-1}\alpha(1-\alpha)^i\mathbf{P}^i\mathbf{D}_v^{-1} + (1-\alpha)^T\mathbf{P}^T\mathbf{D}_v^{-1}\right)\right] \\
&= \operatorname{tlog}^{\circ}\left[\operatorname{vol}(\mathcal{H})\mathbf{D}_v^{-\frac{1}{2}}\mathbf{V}\left(\sum_{i=0}^{T-1}\alpha(1-\alpha)^i\boldsymbol{\Sigma}^{2i}+(1-\alpha)^T\boldsymbol{\Sigma}^{2T}\right)\mathbf{V}^{\mathsf{T}}\mathbf{D}_v^{-\frac{1}{2}}\right].
\end{aligned}$$

Denote $\widehat{\boldsymbol{\Sigma}} = \sum_{i=0}^{T-1}\alpha(1-\alpha)^i\boldsymbol{\Sigma}^{2i} + (1-\alpha)^T\boldsymbol{\Sigma}^{2T}$, and note that $\widehat{\boldsymbol{\Sigma}}$ is an $n \times n$ diagonal matrix. Simplify the above equation, we get

$$\boldsymbol{\Psi}_T = \operatorname{tlog}^{\circ}\left[\operatorname{vol}(\mathcal{H})\mathbf{D}_v^{-\frac{1}{2}}\mathbf{V}\widehat{\boldsymbol{\Sigma}}\mathbf{V}^{\mathsf{T}}\mathbf{D}_v^{-\frac{1}{2}}\right].$$

This can be reformulated as

$$\boldsymbol{\Psi}_T = \operatorname{tlog}^{\circ}\left(\mathbf{F}^{\mathsf{T}}\mathbf{F}\right), \text{ where } \mathbf{F} = \sqrt{\operatorname{vol}(\mathcal{H})}\mathbf{D}_v^{-1/2}\mathbf{V}\widehat{\boldsymbol{\Sigma}}^{1/2}. \tag{4.19}$$

To acquire the HMS-E matrix $\boldsymbol{\Psi}'_T$ for hyperedge embedding, we can reformulate it as follows, via a similar process:

$$\boldsymbol{\Psi}'_T = \operatorname{tlog}^{\circ}\left(\mathbf{F}'^{\mathsf{T}}\mathbf{F}'\right), \text{ where } \mathbf{F}' = \sqrt{\operatorname{vol}(\mathcal{H})}\mathbf{D}_e^{-1/2}\mathbf{W}^{-1/2}\mathbf{U}\widehat{\boldsymbol{\Sigma}}^{1/2}.$$

In this way, both similarity matrices $\boldsymbol{\Psi}_T$ and $\boldsymbol{\Psi}'_T$ can be easily constructed from the RSVD results of $\widetilde{\mathbf{H}}$ via $\mathbf{F}$ and $\mathbf{F}'$, without the need to compute $\boldsymbol{\Pi}$ and $\boldsymbol{\Pi}'$, both of which require expensive and repeated multiplication of transition matrices $\mathbf{P}$ and $\mathbf{P}'$. Thus, we avoid directly materializing $\boldsymbol{\Psi}_T$ or $\boldsymbol{\Psi}'_T$.

4.4.2 HMS-N and HMS-E Approximations

Despite the reformulation, two efficiency issues still remain. First, the reduced SVD on $\widetilde{\mathbf{H}}$ still has a prohibitive $O\left(n(m+n)\right)$ time complexity. To ensure scalability, we opt for the truncated SVD $\widetilde{\mathbf{H}} \approx \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^\top$, which only keeps the r largest singular values and the corresponding singular vectors. In this SVD, $\mathbf{\Sigma}_r = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$ is a diagonal matrix where σ_i is the i-th largest singular value, while $\mathbf{U}_r \in \mathbb{R}^{(m+n)\times r}$ and $\mathbf{V}_r \in \mathbb{R}^{n\times r}$ are the left and right singular vectors. By replacing $\mathbf{\Sigma}$, $\mathbf{U}$ and $\mathbf{V}$ with the truncated SVD results, we can derive $\widehat{\mathbf{\Sigma}}_r$, $\mathbf{F}_r$ and $\mathbf{F}'_r$, providing rank-r approximations for the node and hyperedge similarity matrices, where the error is bounded in Theorem 4.4.1 proved as follows.

Theorem 4.4.1. *With rank-r matrices $\mathbf{F}_r = \sqrt{\mathrm{vol}(\mathcal{H})}\mathbf{D}_v^{-1/2}\mathbf{V}_r\widehat{\mathbf{\Sigma}}_r^{1/2} \in \mathbb{R}^{n\times r}$ and $\mathbf{F}'_r = \sqrt{\mathrm{vol}(\mathcal{H})}\mathbf{D}_e^{-1/2}\mathbf{W}^{-1/2}\mathbf{U}_r\widehat{\mathbf{\Sigma}}_r^{1/2} \in \mathbb{R}^{(m+n)\times r}$, we have the following approximation guarantee for $\mathbf{\Psi}_T$ and $\mathbf{\Psi}'_T$.*

$$\left\|\mathrm{tlog}^\circ\left(\mathbf{F}_r\mathbf{F}_r^\top\right) - \mathbf{\Psi}_T\right\|_F^2 \leq \left[\left\|\mathbf{D}_v^{1/2}\right\|_F^2\left\|\mathbf{D}_v^{-1/2}\right\|_F^2\sum_{i=r+1}^{n}\widehat{\mathbf{\Sigma}}[i,i]\right]^2,$$

$$\left\|\mathrm{tlog}^\circ\left(\mathbf{F}'_r\mathbf{F}'^T_r\right) - \mathbf{\Psi}'_T\right\|_F^2 \leq \left[\left\|\mathbf{D}_v^{1/2}\right\|_F^2\left\|\mathbf{D}_e^{-1/2}\mathbf{W}^{-1/2}\right\|_F^2\sum_{i=r+1}^{n}\widehat{\mathbf{\Sigma}}[i,i]\right]^2.$$

Proof. First, we prove the inequality w.r.t. the approximate HMS-N matrix $\mathbf{\Psi}_T$. Let $\tilde{\mathbf{F}}$ denote $\mathbf{F}\mathbf{F}^\top$, and $\tilde{\mathbf{F}}_r$ denote $\mathbf{F}_r\mathbf{F}_r^\top$. Then, by the definition of $\mathbf{\Psi}_T$, the l.h.s. of the first inequality can be write as $\mathrm{tlog}^\circ\left(\mathbf{F}_r\mathbf{F}_r^\top\right) - \mathrm{tlog}^\circ\left(\mathbf{F}^\top\mathbf{F}\right)$. For ease of exposition, we denote the result of this formula by $\mathbf{A}$. Then the absolute value of an arbitrary element of the matrix $\mathbf{A}$ is

$$\begin{aligned}|\mathbf{A}[i,j]| &= \left|\mathrm{tlog}\left(\tilde{\mathbf{F}}_r[i,j]\right) - \mathrm{tlog}\left(\tilde{\mathbf{F}}[i,j]\right)\right| \\ &= \left|\log\left(\max\left\{\tilde{\mathbf{F}}_r[i,j],1\right\}\right) - \log\left(\max\left\{\tilde{\mathbf{F}}[i,j],1\right\}\right)\right| \\ &\leq \left|\max\left\{\tilde{\mathbf{F}}_r[i,j],1\right\} - \max\left\{\tilde{\mathbf{F}}[i,j],1\right\}\right| \leq \left|\tilde{\mathbf{F}}_r[i,j] - \tilde{\mathbf{F}}[i,j]\right|.\end{aligned} \tag{4.20}$$

The second equality is due to $\mathrm{tlog}^{\circ}(\cdot)$. The first inequality is because of $|\log x - \log y| \leq |x-y|$ for any $x, y \geq 1$. The second inequality is due to $|\max\{x,1\} - \max\{y,1\}| \leq |x-y|$ for any $x, y \in \mathbb{R}$.

Let $\mathbf{\Sigma}_{r+}$ denote the diagonal matrix of the $(r+1)$-th to n-th largest singular values of $\widetilde{\mathbf{H}}$, and the corresponding singular vectors are $\mathbf{U}_{r+} \in \mathbb{R}^{(m+n)\times(n-r)}$ and $\mathbf{V}_{r+} \in \mathbb{R}^{n\times(n-r)}$. Then we have

$$\begin{aligned}
&\left\|\mathrm{tlog}^{\circ}\left(\mathbf{F}_r\mathbf{F}_r^{\top}\right) - \mathbf{\Psi}_T\right\|_F^2 \leq \|\tilde{\mathbf{F}}_r - \tilde{\mathbf{F}}\|_F^2 = \|\mathbf{F}_r\mathbf{F}_r^{\top} - \mathbf{F}\mathbf{F}^{\top}\|_F^2 \\
&= \mathrm{vol}^2(\mathcal{H})\left\|\mathbf{D}_v^{-1/2}\mathbf{V}_r\widehat{\mathbf{\Sigma}}_r\mathbf{V}_r^{\top}\left(\mathbf{D}_v^{-1/2}\right)^{\top} - \mathbf{D}_v^{-1/2}\mathbf{V}\widehat{\mathbf{\Sigma}}\mathbf{V}^{\top}\left(\mathbf{D}_v^{-1/2}\right)^{\top}\right\|_F^2 \\
&= \mathrm{vol}^2(\mathcal{H})\left\|\mathbf{D}_v^{-1/2}\mathbf{V}_{r+}\widehat{\mathbf{\Sigma}}_{r+}\mathbf{V}_{r+}^{\top}\mathbf{D}_v^{-1/2}\right\|_F^2 \leq \mathrm{vol}^2(\mathcal{H})\left[\left\|\mathbf{D}_v^{-1/2}\right\|_F^2\right]^2\left\|\mathbf{V}_{r+}\widehat{\mathbf{\Sigma}}_{r+}\mathbf{V}_{r+}^{\top}\right\|_F^2 \\
&\leq \left[\left\|\mathbf{D}_v^{1/2}\right\|_F^2\left\|\mathbf{D}_v^{-1/2}\right\|_F^2\sum_{i=r+1}^{n}\widehat{\mathbf{\Sigma}}[i,i]\right]^2,
\end{aligned}$$

where the first inequality is due to Ineq. (4.20). The second inequality is due to the property of Frobenius norm. The third inequality follows by rewriting $\mathrm{vol}(\mathcal{H})$ as a Frobenius norm. $\mathbf{V}_{r+}$ and $\mathbf{V}_{r+}^{\top}$ are orthogonal and $\widehat{\mathbf{\Sigma}}_{r+}$ is diagonal. Thus, $\mathbf{V}_{r+}\widehat{\mathbf{\Sigma}}_{r+}\mathbf{V}_{r+}^{\top}$ must be the SVD decomposition of some matrix, and the Frobenius norm of that matrix is the sum of the squared singular values. Then, we apply $\sum_i a_i^2 \leq (\sum_i a_i)^2$ where $a_i \geq 0$. Deriving the upper bound in the second inequality of Theorem 4.4.1 is similar and thus omitted. □

The second challenge arises from the quadratic time and space costs of computing the matrix multiplication $\mathbf{F}_r\mathbf{F}_r^{\top}$ and applying the subsequent element-wise $\mathrm{tlog}^{\circ}(\cdot)$ function. To solve this issue, we employ the polynomial tensor sketch (PTS) technique [39] to approximate $\mathrm{tlog}^{\circ}(\mathbf{F}_r\mathbf{F}_r^{\top})$ with $\Gamma = \mathbf{Y}\mathbf{\Theta}\mathbf{Y}^{\top}$, where $\mathbf{Y} \in \mathbb{R}^{n\times(\tau b+1)}$ contains the tensor sketches and the diagonal $\mathbf{\Theta}$ encodes polynomial coefficients. With PTS, we can efficiently bypass direct matrix materialization. Specifically, we first generate tensor sketches based on polynomial degree τ and sketch dimension b. Then, we derive the approximation through a process involving count-sketch matrices, recursive tensor computations based on fast Fourier transform, and regression-based coefficient estimation with sample size c. To analyze, the PTS method takes only linear time $O(n)$ in total, including $O(\tau nr)$ for count-sketch generation, $O(\tau nb)$ for fast Fourier transform and its inverse, and $O(ncr)$ for fitting $\mathrm{tlog}^{\circ}(\cdot)$ via regression. Moreover, the approximation error of PTS is bounded by Lemma 4.4.2 for our approximation based on the theory in [39].

Lemma 4.4.2. *If* $|\mathrm{tlog}(x) - \sum_{i=0}^{\tau} x^i| \leq \epsilon$ *for some* $\epsilon > 0$ *in a closed interval containing all entries of* $\mathbf{F}_r\mathbf{F}_r^{\top}$*, the PTS* $\mathbf{\Gamma} = \mathbf{Y\Theta Y}^{\top}$ *satisfies* $\mathbb{E}\left\|f^{\circ}(\mathbf{F}_r\mathbf{F}_r^{\top}) - \mathbf{\Gamma}\right\|_F^2 \leq 2n^2\epsilon^2 + \sum_{i=1}^{\tau} \frac{2\tau(2+3^i)(\mathbf{\Theta}[i,i])^2}{b}\left[\sum_{j=1}^{n}\|\mathbf{F}_r[j,:]\|_F^{2i}\right]^2.$

Although $\mathbf{\Gamma} = \mathbf{Y\Theta Y}^{\top}$ resembles the eigendecomposition of $\mathbf{\Gamma}$, we cannot directly use $\mathbf{Y}$ and $\mathbf{\Theta}$ to derive embeddings, since the dimension of $\mathbf{Y}$, $n \times (\tau b + 1)$, does not agree with the k-dimensional embedding space. To obtain a k-dimensional decomposition efficiently, avoiding the quadratic complexity of standard factorization, we apply the Lanczos method for eigendecomposition. This method computes the k leading eigenpairs of $\mathbf{\Gamma}$ by iteratively applying the linear operator $\mathcal{L}(\mathbf{v}) = \mathbf{\Gamma v} = \mathbf{Y}\left(\mathbf{\Theta}\left(\mathbf{Y}^{\top}\mathbf{v}\right)\right)$, which multiplies a vector $\mathbf{v}$ by $\mathbf{\Gamma}$ with complexity linear to n. Consequently, the matrix $\mathbf{\Lambda_\Gamma}$ contains the k largest-magnitude eigenvalues of $\mathbf{\Gamma}$, and $\mathbf{Q_\Gamma}$ comprises their corresponding eigenvectors as columns, yielding a factorization: $\mathbf{Q_\Gamma\Lambda_\Gamma Q_\Gamma^{\top}}$. Finally, we get the node embeddings as

$$\mathbf{Z}_{\mathcal{V}} = \mathbf{Q_\Gamma}\mathbf{\Lambda}_{\mathbf{\Gamma}}^{1/2}. \tag{4.21}$$

Following similar processes with details omitted, we can approximate the hyperedge similarity matrix $\mathrm{tlog}^{\circ}(\mathbf{F}_r'\mathbf{F}_r'^{\top})$ with $\mathbf{\Gamma}'$, decomposed as $\mathbf{Q}_{\mathbf{\Gamma}}'\mathbf{\Lambda}_{\mathbf{\Gamma}}'\mathbf{Q}_{\mathbf{\Gamma}}'^{\top}$, and derive the hyperedge embeddings

$$\mathbf{Z}_{\mathcal{E}} = \mathbf{Q}_{\mathbf{\Gamma}}'\mathbf{\Lambda}_{\mathbf{\Gamma}}'^{1/2}. \tag{4.22}$$

Algorithm 9: SAHE

```
Input: Hyperedge incidence matrix $\mathbf{H}_0 \in \mathbb{R}^{m\times n}$, node attribute matrix
       $\mathbf{X} \in \mathbb{R}^{n\times q}$, embedding dimension $k$, algorithm parameters
       $K, r, T, \alpha, \tau, b, c$.
1  $\mathbf{H}, \mathbf{D}_v, \mathbf{D}_e, \mathbf{W} \leftarrow$ ExtendHG$(\mathbf{H}_0, \mathbf{X}, K)$;
2  $\widetilde{\mathbf{H}} \leftarrow \mathbf{W}^{1/2}\mathbf{D}_e^{-1/2}\mathbf{H}\mathbf{D}_v^{-1/2}$ ;            // Eq. (4.17)
3  $\mathbf{U}_r, \mathbf{\Sigma}_r, \mathbf{V}_r \leftarrow$ TruncatedSVD$\left(\widetilde{\mathbf{H}}, r\right)$;
4  $\widehat{\mathbf{\Sigma}}_r \leftarrow \mathbf{I}_r$;
5  for $i \leftarrow 1, \ldots, T$ do
6      $\widehat{\mathbf{\Sigma}}_r \leftarrow \alpha\mathbf{I}_r + (1-\alpha)\mathbf{\Sigma}_r^2\widehat{\mathbf{\Sigma}}_r$;
7  $\mathbf{F}_r \leftarrow \sqrt{\mathrm{vol}(\mathcal{H})}\mathbf{D}_v^{-1/2}\mathbf{V}_r\widehat{\mathbf{\Sigma}}_r^{1/2}$ ;            // Theorem 4.4.1
8  $\mathbf{Y}, \mathbf{\Theta} \leftarrow$ PTS$(\mathbf{F}_r, \mathrm{tlog}, \tau, b, c)$;
9  Linear operator $\mathcal{L}(\mathbf{v}) = \mathbf{Y}\left(\mathbf{\Theta}\left(\mathbf{Y}^\top\mathbf{v}\right)\right)$;
10 $\mathbf{\Lambda}_\mathbf{\Gamma}, \mathbf{Q}_\mathbf{\Gamma} \leftarrow$ Lanczos$(\mathcal{L}, k)$;            // eigen$\left(\mathbf{Y}\mathbf{\Theta}\mathbf{Y}^\top, k\right)$
11 $\mathbf{Z}_\mathcal{V} \leftarrow \mathbf{Q}_\mathbf{\Gamma}\mathbf{\Lambda}_\mathbf{\Gamma}^{1/2}$ ;            // Eq. (4.21)
12 $\mathbf{F}'_r \leftarrow \sqrt{\mathrm{vol}(\mathcal{H})}\mathbf{D}_e^{-1/2}\mathbf{W}^{-1/2}\mathbf{U}_r\widehat{\mathbf{\Sigma}}_r^{1/2}$;
13 $\mathbf{Y}', \mathbf{\Theta}' \leftarrow$ PTS$\left(\mathbf{F}'_r[1:m+1,:], \mathrm{tlog}, \tau, b, c\right)$;
14 Linear operator $\mathcal{L}'(\mathbf{v}) = \mathbf{Y}'\left(\mathbf{\Theta}'\left(\mathbf{Y}'^\top\mathbf{v}\right)\right)$;
15 $\mathbf{\Lambda}'_\mathbf{\Gamma}, \mathbf{Q}'_\mathbf{\Gamma} \leftarrow$ Lanczos$\left(\mathcal{L}', k\right)$;            // eigen$\left(\mathbf{Y}'\mathbf{\Theta}'\mathbf{Y}'^\top, k\right)$
16 $\mathbf{Z}_\mathcal{E} \leftarrow \mathbf{Q}'_\mathbf{\Gamma}\mathbf{\Lambda}'^{1/2}_\mathbf{\Gamma}$ ;            // Eq. (4.22)
17 return $\mathbf{Z}_\mathcal{V}, \mathbf{Z}_\mathcal{E}$;
```

4.4.3 SAHE Algorithm Details

With the aforementioned techniques, we can derive node and hyperedge embeddings efficiently without materializing dense similarity matrices. The pseudocode of SAHE, our proposed method for attributed hypergraph embedding, is presented in Algorithm 9.

Algorithm. After constructing the attribute-extended hypergraph $\mathcal{H}$ at Line 1 by calling Algorithm 7 (Line 1), we get the incidence matrix $\mathbf{H}$, the degree matrices

$\mathbf{D}_v, \mathbf{D}_e$ and the weight matrix $\mathbf{W}$. Then, we can obtain the normalized hypergraph incidence matrix $\widetilde{\mathbf{H}}$ and decompose it into $\mathbf{U}_r, \mathbf{\Sigma}_r$, and $\mathbf{V}_r$ via the rank-r TruncatedSVD method (Lines 2-3). We calculate $\widehat{\mathbf{\Sigma}}_r$ from the r largest singular values of $\widetilde{\mathbf{H}}$ in Lines 4-6. Then we derive the embeddings for node and hyperedges.

For node embeddings (Lines 9-11), we first derive $\mathbf{F}_r$ by its definition in Theorem 4.4.1, and the node similarity matrix becomes $\operatorname{tlog}^{\circ}\left(\mathbf{F}_r\mathbf{F}_r^{\top}\right)$. To compute this $\operatorname{tlog}^{\circ}\left(\mathbf{F}_r\mathbf{F}_r^{\top}\right)$ function efficiently, we derive its polynomial tensor sketches $\mathbf{Y}$ and $\mathbf{\Theta}$. Finally, the node embeddings $\mathbf{Z}_{\mathcal{V}} = \mathbf{Q}_{\mathbf{\Gamma}}\mathbf{\Lambda}_{\mathbf{\Gamma}}^{1/2}$ are obtained by factorizing $\mathbf{\Gamma} = \mathbf{Y}\mathbf{\Theta}\mathbf{Y}^{\top}$ into $\mathbf{Q}_{\mathbf{\Gamma}}\mathbf{\Lambda}_{\mathbf{\Gamma}}\mathbf{Q}^{\top}$ via the Lanczos technique [67].

Following a similar process, we can derive the hyperedge embeddings $\mathbf{Z}_{\mathcal{V}} = \mathbf{Q}'_{\mathbf{\Gamma}}\mathbf{\Lambda}'^{1/2}_{\mathbf{\Gamma}}$ (Lines 12-16), except that in Line 13 we only generate sketches for the first m rows of $\mathbf{F}_r$, since we are only interested in the embeddings of the original hyperedges.

Complexity. With the above approximation techniques, our proposed `SAHE` algorithm has a much lower complexity than the base method. To analyze, we first consider the basic steps. Specifically, the invocation of `ExtendHG` to derive the matrices in Line 1 takes $O(n\log n + nqK)$ time. The multiplication of matrices in Line 2 costs only linear time, since $\mathbf{W}$, $\mathbf{D}_v$, and $\mathbf{D}_e$ are diagonal and $\mathbf{H}$ is a sparse matrix with $n\bar{d} + nK$ nonzero entries. The `TruncatedSVD` technique in Line 3 involves a bounded number of matrix-vector multiplications on $\widetilde{\mathbf{H}}$, and hence incurs $O(n\bar{d} + nK)$ time complexity. Then, the calculation of $\widehat{\mathbf{\Sigma}}_r$ in Lines 4-6 takes a negligible $O(Tr)$ time. As can be seen, the common steps for node and hyperedge embedding only take linear time in total.

To derive the node embeddings, we compute $\mathbf{F}_r$ in Line 7, which takes $O(nr)$ time. Next, recall from Section 4.4.2 that the approximation via the `PTS` method in Line 8 finishes in linear time $O(n)$. As for the Lanczos method in Lines 9-10, the linear operator $\mathcal{L}(\cdot)$ only takes linear time and the $\mathcal{L}(\cdot)$ operator will only be carried out for constant times. Thus, the Lanczos method also only takes linear time. Finally, the time to obtain node embeddings is $O(n)$. By similar arguments, we can conclude that the time to obtain hyperedge embeddings is $O(m+n)$.

To summarize, the overall time complexity of SAHE is $O(n \log n + n\bar{d} + nq + m)$, or simply $O(n \log n)$ as q and $\bar{d}$ can be considered constant. Moreover, the memory overhead of SAHE is $O(n\bar{d} + nq + m)$, which is linear in the size of the input H, since all involved matrices are either sparse or low-dimensional.

Discussion. SAHE achieves substantial speedup at the cost of approximation errors, compared to Base, which directly computes and factorizes the similarity matrices. Experiments show that on small datasets, the performance of SAHE and Base is similar, though Base is often slightly better. However, Base cannot scale to large datasets, while SAHE consistently outperforms existing methods in efficiency and effectiveness. Hence, the efficiency gain of SAHE is well worth the approximation trade-offs.

4.5 Experiments

After providing the experimental settings in Section 4.5.1, we report the performance of node embedding on node classification task in Section 4.5.2 and on hyperedge link prediction task in Section 4.5.3, and the performance of hyperedge embedding on hyperedge classification task in Section 4.5.4. The efficiency results and experimental analysis are reported in Section 4.5.5 and Section 4.5.6. The implementation of our methods is available at `https://github.com/CyanideCentral/AHNEE`.

4.5.1 Experimental Setup

Datasets. Table 5.2 summarizes the statistics of attributed hypergraphs used in our experiments, including the number of nodes (n) and hyperedges (m), the average node degree ($\bar{d}$), the average hyperedge size ($\bar{\delta}$), the dimension of node attributes

Table 4.2: Dataset statistics.

Dataset	n	m	$\bar{d}$	$\bar{\delta}$	q	ℓ
DBLP-CA	2,591	2,690	2.39	2.31	334	4
Cora-CA	2,708	1,072	1.69	4.28	1,433	7
Cora-CC	2,708	1,579	1.77	3.03	1,433	7
Citeseer	3,312	1,079	1.04	3.20	3,703	6
Mushroom	8,124	298	5.0	136.3	126	2
20News	16,242	100	4.03	654.5	100	4
DBLP	41,302	22,263	2.41	4.45	1,425	6
Recipe	101,585	12,387	25.2	206.9	2,254	8
Amazon	2,268,083	4,285,295	32.2	17.1	1,000	15
MAG-PM	2,353,996	1,082,711	7.34	16.0	1,000	22

Table 4.3: Node classification performance. The best three are in gray with darker shades indicating better performance.

Method	DBLP-CA		Cora-CA		Cora-CC		Citeseer		Mushroom		20News		DBLP		Recipe		Amazon		MAG-PM		Rank
	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	
Hyper2vec	0.446	0.410	0.412	0.365	0.493	0.460	0.311	0.258	-	-	-	-	0.702	0.672	-	-	-	-	-	-	8.9
PANE	0.671	0.651	0.516	0.456	0.508	0.491	0.443	0.399	0.910	0.909	0.566	0.464	0.750	0.734	-	-	-	-	0.378	0.230	6.9
AnECI	0.683	0.661	0.625	0.582	0.453	0.367	0.454	0.399	0.914	0.913	0.694	0.589	-	-	-	-	-	-	-	-	7.3
CONN	0.756	0.744	0.684	0.640	0.637	0.577	0.626	0.563	-	-	-	-	0.828	0.814	-	-	-	-	-	-	6.0
VilLain	0.462	0.439	0.457	0.412	0.484	0.490	0.301	0.272	0.984	0.984	0.730	0.645	0.692	0.657	-	-	-	-	-	-	7.6
AnchorGNN	0.275	0.196	0.239	0.096	0.254	0.095	0.195	0.114	0.854	0.853	0.545	0.429	0.271	0.071	0.379	0.069	0.310	0.032	0.252	0.018	9.2
BiANE	0.705	0.682	0.716	0.683	0.652	0.625	0.644	0.579	0.969	0.969	-	-	0.853	0.843	-	-	-	-	-	-	5.0
TriCL	0.787	0.778	0.702	0.677	0.668	0.646	0.540	0.487	0.978	0.978	0.761	0.722	-	-	-	-	-	-	-	-	5.0
HypeBoy	0.812	0.789	0.725	0.688	0.627	0.584	0.476	0.420	0.970	0.970	-	-	-	-	-	-	-	-	-	-	5.8
NetMF	0.536	0.514	0.518	0.458	0.527	0.513	0.324	0.281	0.987	0.987	0.766	0.733	0.744	0.721	-	-	-	-	-	-	6.2
LightNE	0.545	0.519	0.520	0.469	0.533	0.514	0.342	0.295	0.959	0.959	0.700	0.646	0.733	0.712	0.382	0.099	0.443	0.210	0.603	0.353	6.0
Base	0.836	0.828	0.777	0.754	0.753	0.732	0.693	0.628	0.997	0.997	0.801	0.775	0.898	0.894	-	-	-	-	-	-	2.1
SAHE	0.824	0.816	0.753	0.732	0.742	0.720	0.690	0.622	0.999	0.999	0.786	0.748	0.867	0.859	0.630	0.236	0.718	0.396	0.698	0.451	1.6

(q), and the number of ground-truth class labels (ℓ). DBLP-CA, Cora-CA, Cora-CC, Citeseer, and DBLP are benchmark datasets in [136]. Mushroom and 20News are from [16], and Recipe is from [69]. Amazon and MAG-PM are million-scale from [73]. In DBLP-CA, Cora-CA, and MAG-PM, nodes represent publications, and hyperedges link publications by the same author. In DBLP, nodes are authors, and hyperedges connect co-authors of a publication. Cora-CC and Citeseer are co-citation datasets where hyperedges group publications cited together. Nodes in these datasets have textual attributes from abstracts, with class labels indicating research areas. The Mushroom dataset forms hyperedges by connecting mushrooms (nodes) with the same traits. A mushroom has a one-hot binary attribute vector from categorical features and is labeled as edible or poisonous. The 20News dataset forms hyperedges by shared keywords, using TF-IDF vectors as node attributes and topics as labels. Recipe is a recipe-ingredient hypergraph with bag-of-words attributes from instruction texts and dense hyperedge connections. In Amazon, nodes are products, hyperedges connect products reviewed by the same user, and attributes come from metadata, with categories as labels. Hyperedges lack labels, so we assign each the most frequent node label. For example, an author hyperedge in Cora-CA takes the predominant research area among its publications, while in Amazon, a user hyperedge adopts the most common product category.

Baselines. For *node embedding* evaluation, we compare `SAHE` against 11 baselines in total, including the hypergraph embedding approach `Hyper2vec` [45], and three attributed graph embedding approaches (*i.e.*, `PANE` [142], `AnECI` [79], and `CONN` [114]), which are applied to reduced graphs derived from the clique expansion of the hypergraph. Also, we consider two bipartite graph embedding techniques `AnchorGNN` [128] and `BiANE` [47] that are applied to a bipartite graph where hyperedges are treated as a distinct set of nodes separate from the original nodes. Finally, we further include three self-supervised learning baselines (`VilLain` [65], `TriCL` [64], and `HypeBoy` [60]), with `TriCL` and `HypeBoy` targeted for attributed hypergraph embedding, as well as matrix factorization based approaches on the general graph, `NetMF` [99] and `LightNE` [98]. For *hyperedge embedding* evaluation, we also compare these baselines, among which bipartite graph embedding methods (`AnchorGNN` and `BiANE`) can produce embeddings for two parts as node and hyperedge embeddings, respectively. The remaining methods compute a hyperedge embedding by averaging the node embeddings in the hyperedge. In addition, we also compare `SAHE` with the base method in Section 4.3.4 for effectiveness.

Implementation. On all datasets, `SAHE` and `Base` have the identical parameter settings: $K = 10$, $\beta = 1.0$, $\alpha = 0.1$, $T = 10$. For all datasets, `SAHE` performs approximation with $r = 32$, $\tau = 3$, $b = 128$, and $c = 10$, except Mushroom with $r = 16$. We fix the output node and hyperedge embedding dimension k to 32 for all

Table 4.4: Hyperedge link prediction performance. The best three are in gray with darker shades indicating better performance.

Method	DBLP-CA		Cora-CA		Cora-CC		Citeseer		Mushroom		20News		DBLP		Recipe		Amazon		MAG-PM		Rank
	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	
`Hyper2vec`	0.631	0.712	0.667	0.751	0.715	0.751	0.669	0.684	-	-	-	-	0.704	0.741	-	-	-	-	-	-	8.1
`PANE`	0.687	0.774	0.685	0.765	0.747	0.755	0.685	0.680	0.930	0.974	0.513	0.638	0.723	0.831	-	-	-	-	0.622	0.697	6.1
`AnECI`	0.704	0.797	0.695	0.778	0.753	0.836	0.793	0.890	0.947	0.976	0.615	0.617	-	-	-	-	-	-	-	-	5.4
`CONN`	0.797	0.880	0.655	0.710	0.737	0.835	0.751	0.856	-	-	-	-	0.727	0.814	-	-	-	-	-	-	6.3
`VilLain`	0.638	0.721	0.682	0.729	0.729	0.833	0.659	0.717	0.905	0.971	0.500	0.396	0.698	0.676	-	-	-	-	-	-	7.4
`AnchorGNN`	0.530	0.553	0.512	0.525	0.628	0.688	0.565	0.603	0.693	0.822	0.515	0.403	0.516	0.522	0.506	0.553	0.694	0.773	0.484	0.476	9.1
`BiANE`	0.638	0.599	0.648	0.597	0.751	0.721	0.690	0.647	0.941	0.981	-	-	0.681	0.631	-	-	-	-	-	-	7.9
`TriCL`	0.719	0.808	0.682	0.738	0.727	0.837	0.720	0.824	0.942	0.988	0.615	0.858	-	-	-	-	-	-	-	-	5.7
`HypeBoy`	0.718	0.836	0.740	0.843	0.835	0.924	0.741	0.805	0.937	0.982	-	-	-	-	-	-	-	-	-	-	5.4
`NetMF`	0.659	0.715	0.740	0.793	0.722	0.736	0.643	0.617	0.943	0.988	0.755	0.873	0.755	0.817	-	-	-	-	-	-	5.9
`LightNE`	0.632	0.676	0.675	0.672	0.725	0.839	0.671	0.756	0.954	0.988	0.535	0.658	0.696	0.703	0.642	0.689	0.732	0.820	0.746	0.793	5.8
`Base`	0.785	0.893	0.744	0.815	0.790	0.899	0.783	0.905	0.968	0.996	0.825	0.969	0.811	0.896	-	-	-	-	-	-	2.8
`SAHE`	0.776	0.890	0.766	0.828	0.807	0.902	0.801	0.916	0.989	0.999	0.870	0.956	0.824	0.911	0.763	0.830	0.909	0.965	0.761	0.798	1.4

approaches. The parameters for all tested baselines are configured according to their respective papers. Our method `SAHE`, along with most baselines, is implemented in Python, except for the C++ competitor `LightNE`.

Evaluation. We conduct experimental evaluations on a Linux computer with an Intel Xeon Platinum 8338C CPU, an NVIDIA RTX 3090 GPU, and 384 GB of RAM, where a maximum of 16 CPU threads are available. The methods `AnECI`, `CONN`, `VilLain`, `AnchorGNN`, `TriCL`, and `HypeBoy` benefit from GPU acceleration, while the other methods, including `Base` and `SAHE`, are executed on the CPU. We report average results over 10 repeated runs. If an approach fails to complete within 24 hours or runs out of memory, it is considered to rank last and we record the result as ‘ - ’ in Tables 4.3-4.5.

4.5.2 Node Classification

For attributed hypergraphs, node classification seeks to predict class labels using node embeddings. We split datasets into training and test sets, using a 20%/80% ratio for most, except Amazon and MAG-PM, where 2% is allocated for training due to their size. Ten random splits are generated per dataset, and we report average results. Embeddings, derived without accessing label information, are used to train a simple linear classifier on the training set, with performance evaluated on the test set. Classification effectiveness is assessed via Micro-F1 (MiF1) and Macro-F1 (MaF1) scores, where higher values indicate better performance.

Table 4.3 shows the results, with the top three performances for each dataset highlighted in gray, using darker shades for better performance. The Rank column indicates the average ranking of each method across all metrics. `SAHE` achieves the best overall rank of 1.6, significantly outperforming the next best competitors, `BiANE` and `TriCL`, which rank at 5.0. On large datasets like Amazon and MAG-PM, most competitors fail to return results within time and memory limits. Compared to `Base` from Section 4.3.4, `SAHE`, developed in Section 4.4, is outperformed slightly on small datasets but excels on large ones where `Base` is inefficient. This highlights the strength of `SAHE`'s approximation techniques in maintaining result quality while improving efficiency. For instance, on Cora-CC, `Base` and `SAHE` secure the first and second positions, respectively, outperforming the third-ranked `TriCL` by up to 8.5% in both MiF1 and MaF1. On the DBLP-CA, Cora-CA, Citeseer, Mushroom, 20News, and DBLP datasets, `SAHE` improves over the best competitors by 1.2%, 2.8%, 4.6%, 1.2%, 2.0%, and 1.4% in MiF1, and 2.7%, 4.4%, 4.3%, 1.2%, 1.5%, and 1.6% in MaF1, respectively. On the densely connected Recipe, `SAHE` significantly outperforms the best competitor by 24.8% in MiF1 and 13.7% in MaF1. On the large Amazon and MAG-PM, `SAHE` also surpasses the runner-up with margins up to 27.5% in MiF1 and 18.6% in MaF1 on Amazon. Table 4.3 demonstrates `SAHE`'s excellent performance in node classification, indicating the high quality of node embeddings and the effectiveness of the HMS-N objective from Section 4.3 and algorithm designs in Section 4.4.2.

4.5.3 Hyperedge Link Prediction

Hyperedge link prediction in attributed hypergraphs seeks to identify whether a group of nodes forms a real hyperedge using node embeddings [94, 65]. For each dataset, we divide hyperedges into training and test sets, using an 80%/20% split for smaller

Table 4.5: Hyperedge classification performance. The best three are in gray with darker shades indicating better performance. (20News is excluded for lack of suitable labels.)

Method	DBLP-CA		Cora-CA		Cora-CC		Citeseer		Mushroom		DBLP		Recipe		Amazon		MAG-PM		Rank
	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	
Hyper2vec	0.569	0.518	0.439	0.370	0.794	0.786	0.589	0.511	-	-	0.599	0.553	-	-	-	-	-	-	7.7
PANE	0.704	0.673	0.515	0.426	0.737	0.725	0.567	0.495	0.769	0.765	0.751	0.731	-	-	-	-	0.303	0.111	6.4
AnECI	0.685	0.664	0.599	0.529	0.542	0.484	0.534	0.398	0.821	0.813	-	-	-	-	-	-	-	-	7.6
CONN	0.809	0.786	0.641	0.587	0.781	0.760	0.689	0.612	-	-	0.837	0.815	-	-	-	-	-	-	5.6
VilLain	0.567	0.515	0.459	0.382	0.799	0.790	0.619	0.538	0.838	0.835	0.550	0.486	-	-	-	-	-	-	6.4
AnchorGNN	0.307	0.251	0.185	0.142	0.194	0.146	0.201	0.170	0.622	0.602	0.267	0.087	0.454	0.078	0.372	0.036	0.334	0.044	9.1
BiANE	0.479	0.408	0.241	0.179	0.577	0.506	0.462	0.377	0.767	0.762	0.462	0.342	-	-	-	-	-	-	8.8
TriCL	0.804	0.778	0.646	0.590	0.820	0.808	0.659	0.579	0.838	0.834	-	-	-	-	-	-	-	-	5.2
HypeBoy	0.820	0.799	0.719	0.662	0.794	0.775	0.728	0.630	0.835	0.830	-	-	-	-	-	-	-	-	5.3
NetMF	0.650	0.607	0.452	0.399	0.797	0.792	0.596	0.530	0.879	0.877	0.730	0.699	-	-	-	-	-	-	5.8
LightNE	0.654	0.610	0.458	0.399	0.800	0.791	0.617	0.534	0.847	0.844	0.702	0.673	0.199	0.051	0.774	0.485	0.502	0.179	4.9
Base	0.858	0.838	0.775	0.740	0.852	0.846	0.770	0.684	0.926	0.925	0.908	0.898	-	-	-	-	-	-	2.1
SAHE	0.854	0.836	0.764	0.711	0.850	0.839	0.756	0.669	0.908	0.907	0.863	0.843	0.668	0.236	0.823	0.429	0.755	0.470	1.7

datasets and 98%/2% for larger ones like Amazon and MAG-PM. For each real hyperedge, we create a negative counterpart by randomly selecting nodes to match its size. Node embeddings are derived from the training set's real hyperedges and full attribute data, excluding test hyperedges and labels. A linear binary classifier is trained to differentiate real from negative hyperedges, using max-min aggregation of node embeddings as input. This model is tested on the test set, predicting real and negative hyperedges. We repeat this process over 10 random splits, averaging the results. Performance is assessed by accuracy (Acc) and area under the ROC curve (AUC), with higher scores indicating better performance.

Table 4.4 shows that `SAHE` ranks highest overall with a score of 1.4, significantly outperforming the strongest baseline, `HypeBoy`, which has a rank of 5.4. While `Base` performs well on smaller datasets, it struggles with larger ones. In contrast, `SAHE` maintains top performance on large datasets like Amazon and MAG-PM, where other methods falter. For instance, on Recipe, `SAHE` exceeds `LightNE` by 12.1% in accuracy and 14.1% in AUC. On the large Amazon dataset, `SAHE` achieves 90.9% accuracy and 96.5% AUC, improving by 17.7% and 14.5% over the runner-up, `LightNE`, which scores 73.2% accuracy and 82.0% AUC. These results confirm that `SAHE` generates high-quality node embeddings, validating the effectiveness of our proposed node similarity measure and embedding objective.

4.5.4 Hyperedge Classification

We evaluate hyperedge embeddings using a classification task that predicts a hyperedge's label from its embedding vector. Hyperedges are split into training and test sets with a 20%/80% ratio, except for Amazon and MAG-PM, which use a 2%/98% split due to their size. Embeddings are computed from the attributed hypergraph without label information. A linear classifier is trained on the training set, using hyperedge embeddings as input and their labels as targets. Performance is assessed on the test set, averaged over 10 random splits, and measured by MiF1 and MaF1.

Table 4.5 shows that `SAHE` ranks first overall with a score of 1.7, significantly outperforming the closest competitor, `LightNE`, which ranks at 4.9. On large datasets like Amazon and MAG-PM, most competitors fail due to time or memory constraints. Unlike `Base`, which struggles with larger datasets, `SAHE` excels on both small and large datasets, thanks to its efficient approximation techniques in Section 4.4. Compared to the runner-up baseline, `SAHE` improves MiF1 by 4.5% and MaF1 by 4.9% on Cora-CA, and by large margins of 21.4% in MiF1 and 15.8% in MaF1 on the Recipe dataset. On the large MAG-PM dataset, `SAHE` outperforms `LightNE` by 25.3% in MiF1 and 29.1% in MaF1. This suggests that simply averaging node embeddings, as in baseline methods, is insufficient for hyperedge embedding. The performance of `SAHE` shows the effectiveness of the HMS-E similarity objective and approximation techniques in Sections 4.3.3 and 4.4.

4.5.5 Embedding Efficiency

Figure 4.5 reports the time of all methods to generate node and hyperedge embeddings across the 8 datasets, with each chart's y-axis showing running time in seconds on a logarithmic scale and stars marking the top-performing competitors in all three tasks.

Observe that (i) `SAHE` consistently outperforms all competitors in terms of efficiency

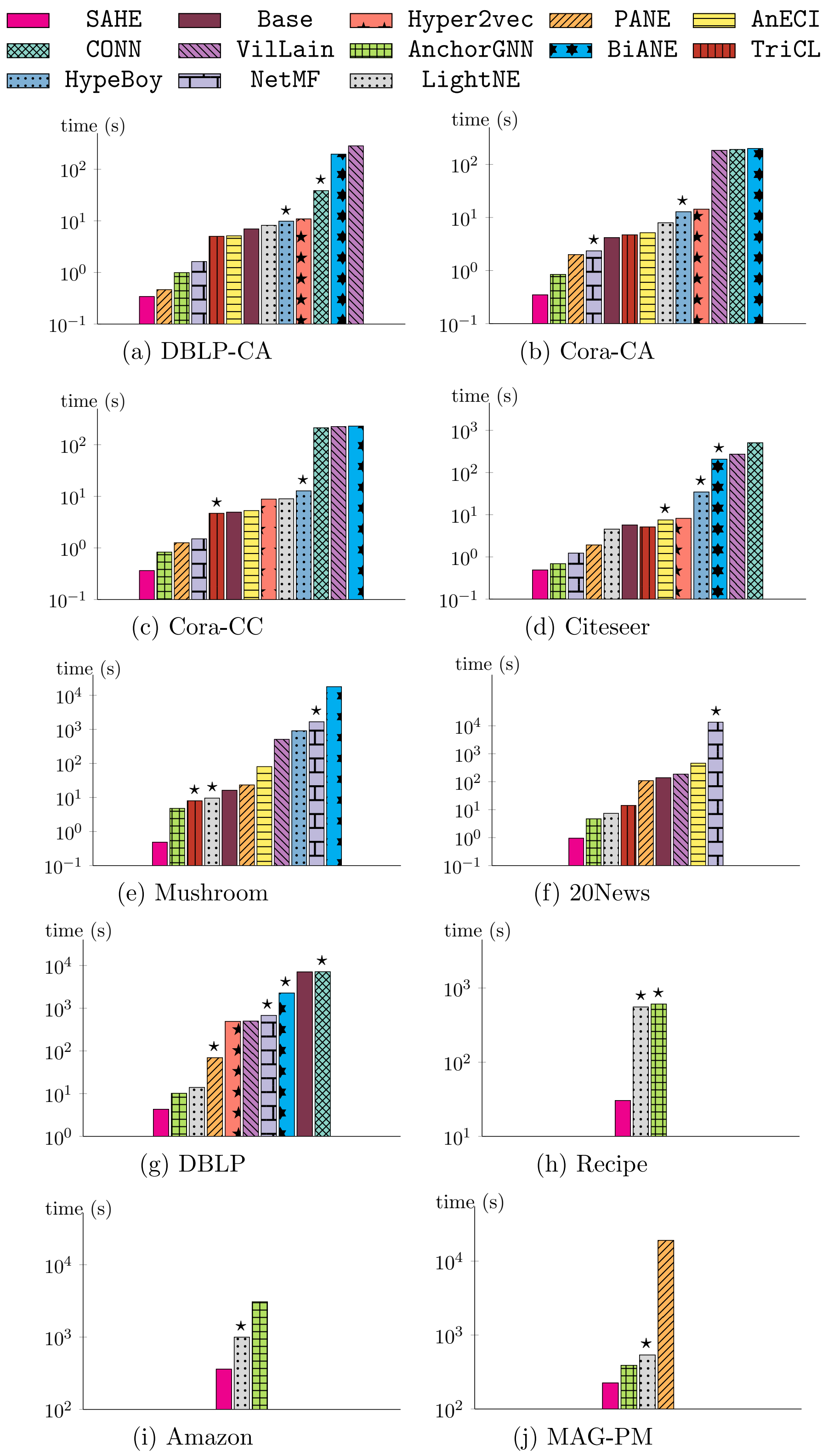


Figure 4.5: Running time of generating node and hyperedge embeddings together in seconds (⋆ marks the best competitors in Tables 4.3, 4.4, 4.5).

across all datasets, regardless of the competitors' quality; more importantly, (ii) `SAHE` demonstrates a significant speed advantage over the most effective competitors marked by stars in Figure 4.5, often being faster by orders of magnitude. Taking the Citeseer dataset as example, Figure 4.5d shows that `SAHE` is 422.5× faster than `BiANE` (3rd place in node classification), 15.3× faster than `AnECI` (3rd place in hyperedge link prediction), and 70.2× faster than `HypeBoy` (3rd place in hyperedge classification), while `SAHE` outperforms all baselines in these tasks. In Figure 4.5f, `SAHE` takes just 0.951 seconds compared to 13,536 seconds for `NetMF`, the runner-up in embedding quality. This is because `NetMF` operates on a dense clique-expansion graph reduced from the hyperedges, which incurs a quadratic complexity for the factorization-based algorithm. On the million-scale Amazon dataset, `SAHE` is much faster than the competitors, such as `LightNE`, with an average rank of 6.0, compared to the 1.4 average rank of `SAHE` in Table 4.3. These results underscore the combination of high-quality embeddings and excellent efficiency achieved by `SAHE`.

4.5.6 Experimental Analysis

Scalability test. We assess scalability on synthetic attributed hypergraphs with the number of nodes n ranging from 2 to 10 million. Each hypergraph is generated as a 3-uniform hypergraph with n hyperedges of size 3 [33], and each node is assigned 100 random binary attributes. Figure 4.6 shows the time and memory usage of `SAHE` against the scalable baseline `LightNE` on CPU. `SAHE` exhibits near-linear scalability and outperforms `LightNE` in both metrics, confirming the complexity analysis in Section 4.4.3 and demonstrating the efficiency of `SAHE` for large datasets in practice.

Approximation error. `SAHE` improves efficiency by introducing acceptable approximation errors compared to `Base`, which directly computes and factorizes similarity matrices. Table 4.6 quantifies this loss by reporting the mean absolute error (MAE)

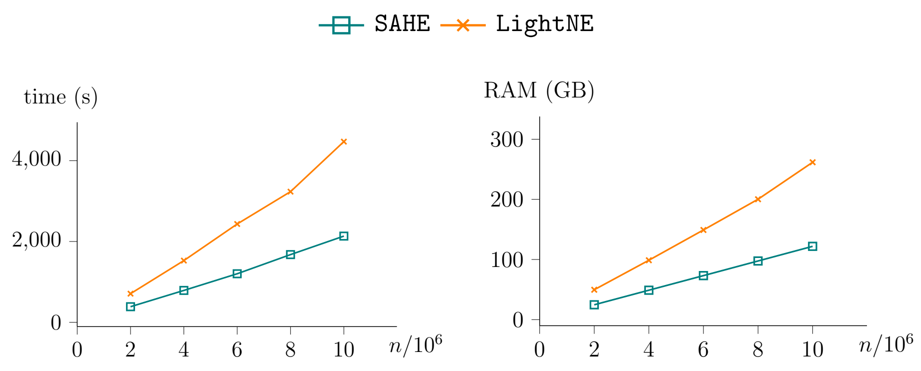


Figure 4.6: Scalability Test.

Table 4.6: Approximation Error (MAE).

Dataset	HMS-N		HMS-E	
	Base	SAHE	Base	SAHE
DBLP-CA	0.0887	0.1281	0.0795	0.2141
Cora-CA	0.0965	0.1384	0.0770	0.2761
Cora-CC	0.0970	0.1546	0.0551	0.1714
Citeseer	0.0927	0.1446	0.0629	0.2096

between normalized HMS-N and HMS-E matrices and their embedding dot product matrices. Specifically, similarity matrices are normalized by their diagonal mean to align self-similarity scales, and MAE is computed as the difference between the embedding dot product and the similarity matrices. Results show low errors for both methods, with Base achieving slightly lower MAE. This confirms SAHE effectively approximates similarity measures with small errors, enabling comparable effectiveness while ensuring efficiency.

Varying K. Figure 4.7 shows the MiF1 for node classification, Acc for hyperedge link prediction, and the time to construct attribute-based hyperedges in $\mathcal{E}_K$ for the Cora-CA and Amazon datasets. As K varies from 2 to 100, time costs rise significantly, especially for $K > 20$. Embedding quality improves notably as K increases from 2 to 10, highlighting the importance of incorporating attribute similarity. However, beyond $K = 10$, the metrics stabilize and then decline, likely due to the inclusion of nodes with dissimilar attributes, which introduces noise. Thus, we set parameter K to 10 over all datasets.

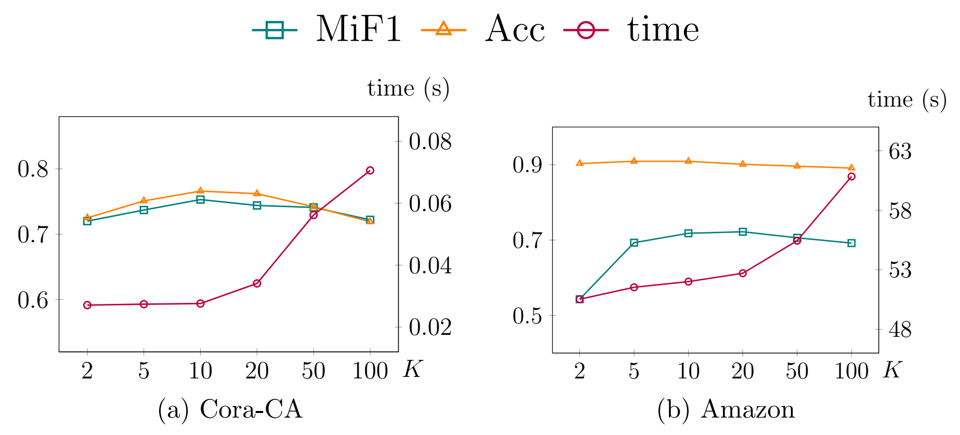


Figure 4.7: Varying K for SAHE.

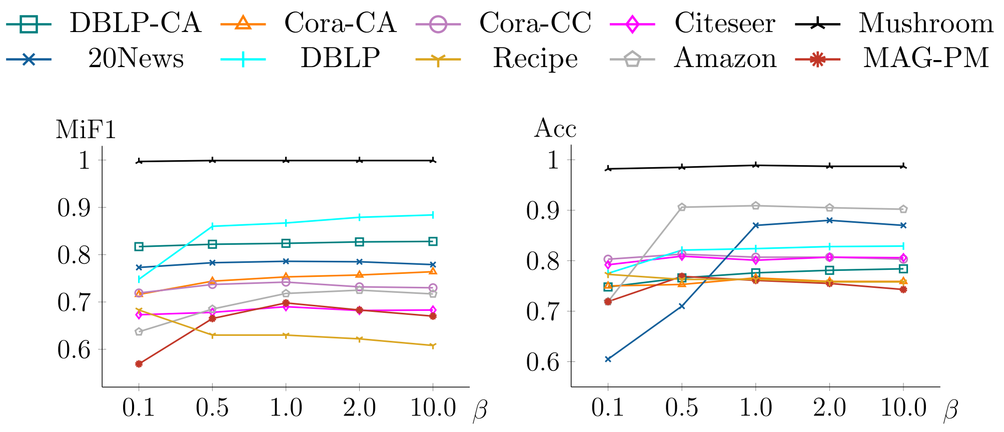


Figure 4.8: Varying β for SAHE.

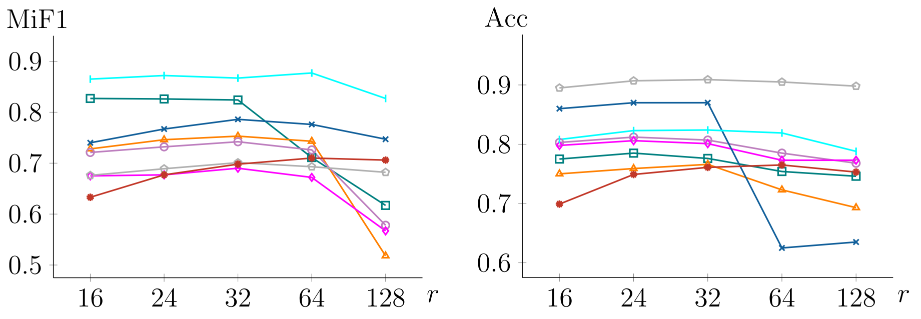


Figure 4.9: Varying r for SAHE.

Varying β. Parameter β balances between attribute-based hyperedges $\mathcal{E}_K$ and original hyperedges $\mathcal{E}$ in $\mathcal{H}$ in Section 4.3.1. Figure 4.8 shows that as β increases from 0.1 to 1, micro-F1 for node embedding generally increase. Beyond 1.0, scores stabilize

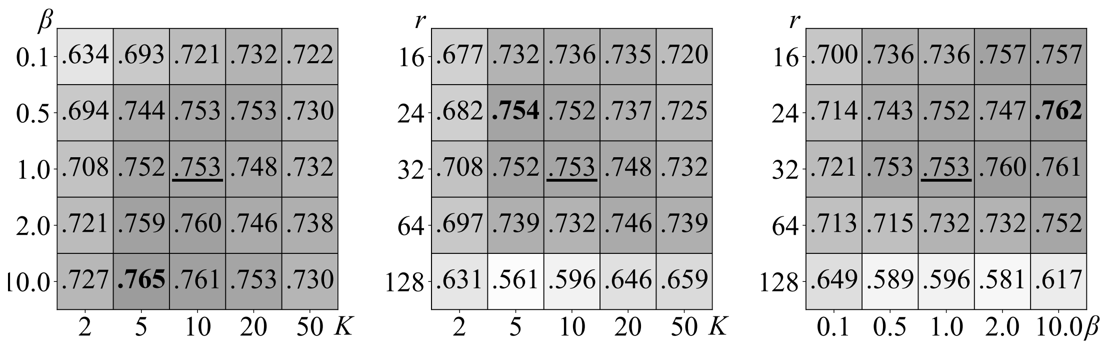


Figure 4.10: Heatmaps between K, β, and r on Cora-CA for node classification. Darker shades indicate higher MiF1. Underlined results are reported in Table 4.3, while the optimal parameter combinations are in bold.

on most datasets, but decline for Cora-CC and MAG-PM when β reaches 10.0. The accuracy of hyperedge link prediction (Acc) increases on 20News and Amazon when β varies from 0.1 to 1.0, then remains stable. Therefore, we set $\beta = 1$ by default.

Varying r. The parameter r represents the dimension of truncated SVD used for approximating HMS-N and HMS-E in Section 4.4.2. We vary r for node classification (MiF1) and hyperedge link prediction (Acc), with results in Figure 4.9. Increasing r from 16 to 32 generally improves or stabilizes both metrics, except for Mushroom, where Acc drops after 24. Beyond 32, scores typically decline. Thus, we set $r = 32$ by default and $r = 16$ for the Mushroom dataset.

Parameter interaction. We analyze the interaction between parameters K, β, and r using node classification on Cora-CA, shown in Figure 4.10. The heatmaps display Micro-F1 scores across parameter combinations. `SAHE` consistently delivers high-quality embeddings, indicated by darker gray levels, confirming robustness to parameter variations. Underlined results are those reported in Table 4.3, while bold values represent optimal performance with fine-tuned parameters, surpassing defaults. This shows `SAHE` delivers strong performance with default settings, without extensive tuning, and provides guidance for adjusting parameters in practice.

Ablation study. To validate our proposed similarity measure formulation, we eval-

Table 4.7: Ablation analysis of HMS-N on node classification performance.

Method	DBLP-CA		Cora-CA		Cora-CC		Citeseer		Mushroom		20News		DBLP	
	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1
HMS-N-no-$\mathcal{E}_K$	0.527	0.492	0.507	0.462	0.559	0.530	0.329	0.278	0.990	0.990	0.796	0.769	0.755	0.733
HMS-N-1-hop	0.811	0.803	0.741	0.720	0.716	0.696	0.680	0.618	0.988	0.988	0.783	0.755	0.852	0.843
HMS-N	**0.836**	**0.828**	**0.777**	**0.754**	**0.753**	**0.732**	**0.693**	**0.628**	**0.997**	**0.997**	**0.801**	**0.775**	**0.898**	**0.894**

Table 4.8: Ablation analysis of HMS-E on hyperedge classification performance.

Method	DBLP-CA		Cora-CA		Cora-CC		Citeseer		Mushroom		DBLP	
	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1
HMS-E-1-hop	0.648	0.605	0.489	0.423	0.765	0.749	0.610	0.521	0.899	0.897	0.650	0.552
HMS-E-no-$\mathcal{E}_K$	0.658	0.615	0.487	0.418	0.821	0.810	0.623	0.549	0.921	0.920	0.747	0.716
HMS-E	**0.858**	**0.838**	**0.775**	**0.740**	**0.852**	**0.846**	**0.770**	**0.684**	**0.926**	**0.925**	**0.908**	**0.898**

Table 4.9: Node classification performance for extended baselines.

Method	DBLP-CA		Cora-CA		Cora-CC		Citeseer		Mushroom		20News		DBLP		Recipe		Amazon		MAG-PM		Rank
	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	
`Hyper2vec+`	0.772	0.763	0.696	0.663	0.643	0.615	0.629	0.574	0.935	0.935	0.776	0.734	0.863	0.858	-	-	-	-	-	-	2.8
`Hyper2vec`	0.446	0.410	0.412	0.365	0.493	0.460	0.311	0.258	-	-	-	-	0.702	0.672	-	-	-	-	-	-	3.8
`TriCL+`	0.797	0.788	0.709	0.678	0.648	0.629	0.616	0.559	0.983	0.983	0.622	0.567	-	-	-	-	-	-	-	-	2.6
`TriCL`	0.787	0.778	0.702	0.677	0.668	0.646	0.540	0.487	0.978	0.978	0.761	0.722	-	-	-	-	-	-	-	-	2.8
`SAHE`	0.824	0.816	0.753	0.732	0.742	0.720	0.690	0.622	0.999	0.999	0.786	0.748	0.867	0.859	0.630	0.236	0.718	0.396	0.698	0.451	**1.0**

Table 4.10: Hyperedge link prediction performance for extended baselines.

Method	DBLP-CA		Cora-CA		Cora-CC		Citeseer		Mushroom		20News		DBLP		Recipe		Amazon		MAG-PM		Rank
	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	Acc	AUC	
`Hyper2vec+`	0.735	0.821	0.613	0.668	0.699	0.795	0.712	0.803	0.932	0.980	0.622	0.749	0.672	0.747	-	-	-	-	-	-	3.1
`Hyper2vec`	0.631	0.712	0.667	0.751	0.715	0.751	0.669	0.684	-	-	-	-	0.704	0.741	-	-	-	-	-	-	3.6
`TriCL+`	0.747	0.881	0.721	0.812	0.767	0.912	0.776	0.900	0.933	0.962	0.523	0.525	-	-	-	-	-	-	-	-	2.5
`TriCL`	0.719	0.808	0.682	0.738	0.727	0.837	0.720	0.824	0.942	0.988	0.615	0.858	-	-	-	-	-	-	-	-	2.8
`SAHE`	0.776	0.890	0.766	0.828	0.807	0.902	0.801	0.916	0.989	0.999	0.870	0.956	0.824	0.911	0.763	0.830	0.909	0.965	0.761	0.798	**1.1**

Table 4.11: Hyperedge classification performance for extended baselines.

Method	DBLP-CA		Cora-CA		Cora-CC		Citeseer		Mushroom		DBLP		Recipe		Amazon		MAG-PM		Rank
	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	
`Hyper2vec+`	0.822	0.798	0.702	0.654	0.804	0.799	0.717	0.637	0.901	0.900	0.851	0.833	-	-	-	-	-	-	2.2
`Hyper2vec`	0.569	0.518	0.439	0.370	0.794	0.786	0.589	0.511	-	-	0.599	0.553	-	-	-	-	-	-	3.8
`TriCL+`	0.803	0.775	0.646	0.593	0.824	0.809	0.670	0.596	0.831	0.828	-	-	-	-	-	-	-	-	2.9
`TriCL`	0.804	0.778	0.646	0.590	0.820	0.808	0.659	0.579	0.838	0.834	-	-	-	-	-	-	-	-	2.9
`SAHE`	0.854	0.836	0.764	0.711	0.850	0.839	0.756	0.669	0.908	0.907	0.863	0.843	0.668	0.236	0.823	0.429	0.755	0.470	**1.0**

uate two ablated versions: HMS-N/HMS-E-no-$\mathcal{E}_K$, which excludes attribute-based hyperedges, and HMS-N/HMS-E-1-hop, which restricts random walks to a single hop. Node and hyperedge embeddings derived from these similarity matrices are assessed via the classification task, with results in Tables 4.7-4.8. The full HMS-N and HMS-E measures generally outperform both ablated versions, confirming the effectiveness of our approach.

Adapting HMS-N and HMS-E to existing methods. We explore two strategies to integrate the key intuitions of HMS-N and HMS-E into existing methods, evaluating their impact on embedding quality alongside SAHE. First, we adapt our multi-hop random walk process into the Hyper2vec framework, yielding Hyper2vec+ for node and hyperedge embeddings. Second, for graph neural network models that do not use random walks, such as TriCL, we enhance it by concatenating the HMS-N matrix with node features, resulting in TriCL+, where hyperedge embeddings are derived by averaging node embeddings. Tables 4.9-4.11 summarize the results for node classification, hyperedge link prediction, and hyperedge classification tasks, respectively. The last column shows the average rank of each method across all datasets, with lower ranks indicating better overall performance. The extended baselines benefit from incorporating our similarity measures to varying extents. For instance, Hyper2vec+ outperforms Hyper2vec, while TriCL+ shows occasional improvements over TriCL. Besides, Hyper2vec+ integrates our random walk scheme, making it more efficient than Hyper2vec's second-order random walks, enabling it to process datasets like 20News and Mushroom within the 24-hour limit. These results highlight that the impact of our ideas depends on the baseline design. Overall, SAHE achieves the best overall rank, consistently delivering superior performance across diverse settings and scaling to large datasets that other methods cannot handle.

Varying α. Figure 4.11 shows MiF1 for node classification in (a) and Acc for hyperedge link prediction in (b) as α varies from 0.001 to 0.3. SAHE demonstrates consistent performance across most values, except for the very small $\alpha = 0.001$. This confirms SAHE's robustness to different α values, with $\alpha = 0.1$ chosen as the default setting.

Varying τ and b. Parameters τ and b balance approximation accuracy and efficiency, with larger values improving accuracy at higher computational cost. Figure 4.12(a) and (b) show MiF1 for node classification and running time as τ varies from 1 to 9, while Figure 4.13 depicts results for b ranging from 16 to 256. MiF1 initially improves and then stabilizes as these parameters increase, while running time continues to rise. The default settings of $\tau = 3$ and $b = 128$ effectively balance quality and efficiency, and performance remains robust across a range of values.

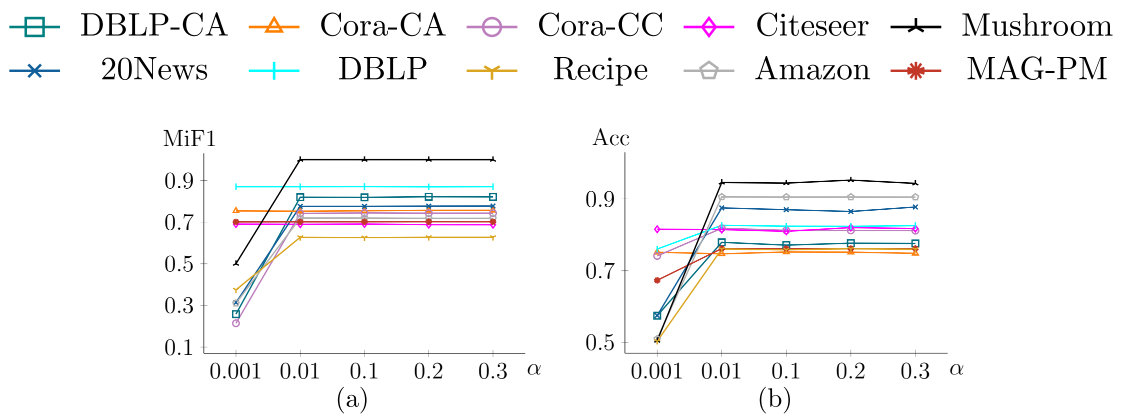


Figure 4.11: Varying α.

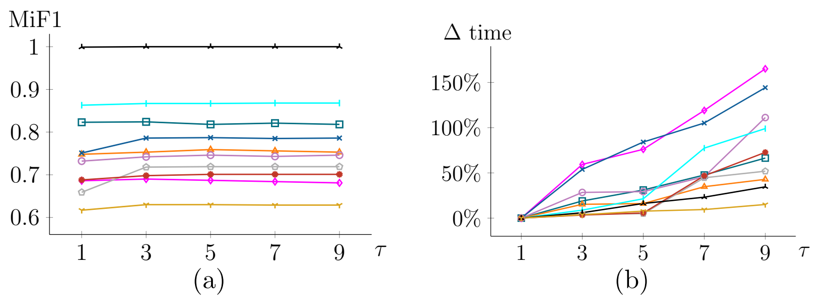


Figure 4.12: Varying τ.

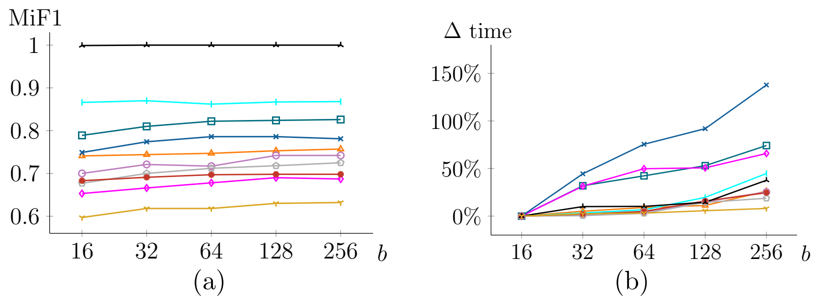


Figure 4.13: Varying b.

4.6 Summary

This chapter presents `SAHE`, an efficient algorithm for the Attributed Hypergraph Embedding (AHE) problem. `SAHE` generates node and hyperedge embeddings that preserve both higher-order connectivity among nodes and collective attribute-based similarities among hyperedges. By introducing multi-hop similarity measures and leveraging optimized decomposition techniques, SAHE achieves log-linear time complexity. Extensive evaluation across 10 real-world datasets shows that SAHE consistently outperforms 11 baseline methods in both scalability and embedding quality, demonstrating its practical effectiveness for attributed hypergraph representation.

Chapter 5

SGLA: Multi-view Attributed Graph Integration

This chapter presents `SGLA` and `SGLA+` [70], spectrum-guided methods for clustering and embedding in multi-view attributed graphs, advancing the thesis's goal of developing effective and scalable solutions for attributed network structures. Unlike Chapters 3 and 4, which focus on networks with a single attribute view and primarily one network view, this work tackles the challenge of integrating multiple graph and attribute views. Together, these contributions enhance the analysis of diverse attributed network structures.

5.1 Introduction

A *multi-view attributed graph* (MVAG) describes a set of entities with multiple *graph views* and *attribute views*, illustrating their relationships and properties from various perspectives or data sources. For example, regarding a group of people, one graph view may focus on their social relations on Facebook, whereas another graph view may represent their business connections on LinkedIn. Moreover, attribute views

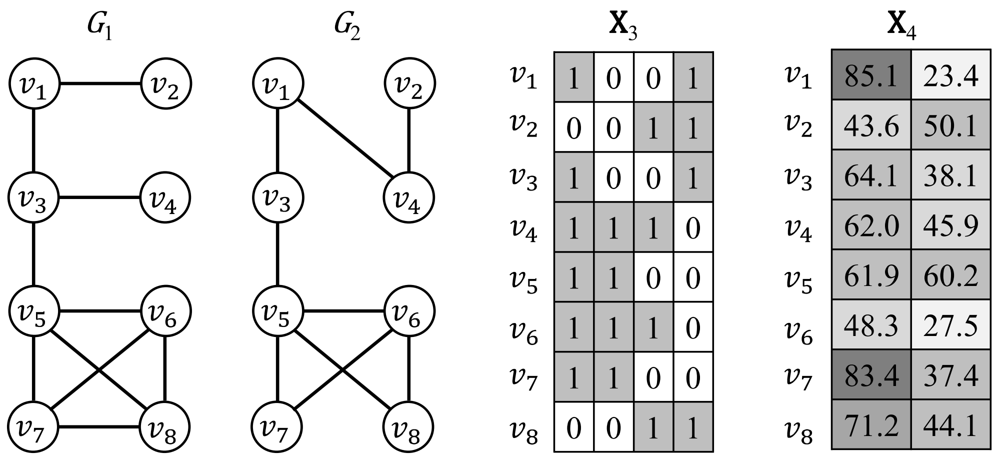

G_1
G_2
$\mathbf{X}_3$
$\mathbf{X}_4$
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8
v_1: 1 0 0 1 | 85.1 23.4
v_2: 0 0 1 1 | 43.6 50.1
v_3: 1 0 0 1 | 64.1 38.1
v_4: 1 1 1 0 | 62.0 45.9
v_5: 1 1 0 0 | 61.9 60.2
v_6: 1 1 1 0 | 48.3 27.5
v_7: 1 1 0 0 | 83.4 37.4
v_8: 0 0 1 1 | 71.2 44.1

Figure 5.1: Multi-view attributed graph $\mathcal{G}$ with two graph views G_1 and G_2, and two attribute views $\mathbf{X}_3$ and $\mathbf{X}_4$ of categorical and numerical attributes respectively.

may comprise diverse numerical, categorical, or visual features. Graph analytics for MVAGs, especially clustering and embedding, are of particular interest as they find important applications. For instance, clustering on MVAGs constructed from visual descriptors is effective for neuroimaging analysis of diseases [153]. MVAG embeddings are useful in recommendation systems in e-commerce [119], spam detection on social networks [68], and predicting drug-disease associations in bioinformatics [32]. Figure 5.1 presents a minimal example of an MVAG of 8 entities, described by 2 graph views and 2 attribute views.

It is crucial but challenging to manage complex MVAGs to holistically utilize the graph views and attribute views for the clustering and embedding tasks, particularly with large MVAGs. In an MVAG $\mathcal{G}$, different graph views can display varying topological structures, and the graph and attribute views represent distinct data models that cannot be directly integrated. Moreover, the considerable data volumes typical in real-world applications pose a significant challenge to efficiency and scalability. These challenges hinder the effective management of MVAG data in an efficient manner.

As reviewed in Section 2.3, an array of approaches [24, 79, 152, 142, 134] are only designed for attributed graphs with one graph view or one attribute view. Other methods [160, 25] handle attribute views without considering graphs, and thus they often yield suboptimal results for MVAGs in the experiments. Existing methods specialized for MVAG clustering or embedding are often built upon sophisticated graph neural network operations [14, 86], and consequently struggle with efficiency and scalability. Several clustering methods [91, 77] attempt to learn a graph structure that aligns with all views in $\mathcal{G}$, requiring a huge number of variables to be solved. Summing up, existing methods either produce subpar results or require excessive computational resources to manage large-scale MVAGs.

In this chapter, we focus on the important problem of how to utilize the rich semantics of all views in an MVAG $\mathcal{G}$, and develop an effective and efficient Spectrum-Guided Laplacian Aggregation approach [70], exploiting the intrinsic spectral properties to cohesively integrate all views of $\mathcal{G}$ into an MVAG Laplacian matrix $\mathcal{L}$ for clustering and embedding.

Our main designs include a carefully formulated objective for the integration (Section 5.3), an efficient method `SGLA` producing high-quality results (Section 5.4.1), and `SGLA`+ that boosts efficiency further while maintaining the effectiveness (Section 5.4.2). Specifically, our strategy is to perform a weighted aggregation of normalized Laplacian matrices from all views in $\mathcal{G}$ to produce the integrated Laplacian $\mathcal{L}$. However, a critical challenge is choosing appropriate view weights to produce an effective $\mathcal{L}$ that preserves the fundamental characteristics of $\mathcal{G}$, *i.e.*, community structure and node connectivity, which are important for clustering and embedding. With this in mind, we design an objective function on the basis of spectral graph theory. In particular, we align the spectrum of $\mathcal{L}$ with a normalized-cut community measure and a graph conductance measure, and propose eigengap and connectivity objectives accordingly. The overall objective combines the eigengap and connectivity objectives, assisted with a regularization term, to determine the appropriate weights of each view in $\mathcal{G}$ to get $\mathcal{L}$. It is challenging to optimize the overall objective in search of appropriate view weights, since it is infeasible to exhaust all weight combinations, and the objective evaluation is computationally expensive. To mitigate the challenges, in Section 5.4, we develop the `SGLA` algorithm to find a desirable solution, which already achieves excellent performance compared with existing methods. Nevertheless, `SGLA` needs to evaluate the objective at every iteration, causing significant overhead. To boost efficiency with fewer objective evaluations, we develop the `SGLA`+ algorithm, which employs sampling and interpolation approximation to quickly find an effective solution. In our experiments, we couple `SGLA` and `SGLA`+ with spectral clustering and embedding methods, to compare them against 12 clustering baselines and 8 embedding baselines over 8 real-world MVAG datasets. `SGLA` and `SGLA`+ achieve better performance in terms of both effectiveness and efficiency. For instance, on large-scale datasets, such as MAG-phy with 4 views and 2.35 million nodes, our methods efficiently produce high-quality results, while most baselines fail to scale.

The contributions of this work are summarized as follows:

- We propose an efficient and effective spectrum-guided aggregation scheme for MVAG clustering and embedding.

- We derive a novel objective formulation, consisting of the eigengap and connectivity objectives, to find appropriate view weights that preserve the community structure and node connectivity in MVAGs.

- We develop two efficient algorithms, `SGLA` and `SGLA`+, with several speedup techniques to optimize the objective.

- Extensive experiments on 8 real-world MVAGs validate the superior performance of `SGLA` and `SGLA`+.

5.2 Preliminaries and Problem Statement

5.2.1 Preliminaries

Multi-view Attributed Graph. Figure 5.1 shows an MVAG of 8 nodes with 4 views, including 2 graph views and 2 attribute views. Graph views G_1 and G_2 are two simple graphs, while attribute views $\mathbf{X}_3$ and $\mathbf{X}_4$ are binary and numerical attributes, respectively. Formally, we denote an MVAG $\mathcal{G}$ with p graph views and q attribute views by $\mathcal{G} = \{\mathcal{V}, \mathcal{E}_1, \dots, \mathcal{E}_p, \mathbf{X}_{p+1}, \dots, \mathbf{X}_{p+q}\}$, where $\mathcal{V}$ is the set of n nodes, $\mathcal{E}_i$ is the set of edges in the i-th graph view $G_i = \{\mathcal{V}, \mathcal{E}_i\}$ that is a simple graph, and matrix $\mathbf{X}_{p+j}$ contains the values in the j-th attribute view. We focus on MVAGs with a total of $r = p+q > 2$ views. Denote the number of edges in G_i by m_i, and the total number of edges in $\mathcal{G}$ as $m = \sum_{i=1}^{p} m_i$. G_i has an adjacency matrix $\mathbf{A}_i \in \mathbb{R}^{n \times n}$, and a node v_a in G_i has a generalized degree $\delta_i(v_a)$ equal to the total weight of incident edges.

Normalized Laplacian. The normalized Laplacian of a simple graph G is $\mathbf{L}(G) = \mathbf{I}_n - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, where $\mathbf{D}$ is the diagonal degree matrix and $\mathbf{I}_n$ is the identity matrix [18, 112].

MVAG Clustering and Embedding. The two analytic tasks for MVAGs are stated as follows:

- *Clustering* is to divide the nodes in $\mathcal{G}$ into k disjoint non-empty subsets $\{\mathcal{C}_1, \dots, \mathcal{C}_k\}$, *i.e.*, k clusters, such that nodes within each cluster tend to form dense connections in graph views and share similar values in attribute views.
- *Embedding* is to map each node $\mathcal{G}$ to a low-dimensional embedding vector that captures its features inherent to the graph views and attribute views in $\mathcal{G}$.

Table 5.1 lists the frequently used notations.

5.2.2 Problem Statement

Given an MVAG $\mathcal{G}$, our goal is to generate an MVAG Laplacian matrix $\mathcal{L}$ as the multi-view integration, which empowers classic methods to handle clustering and embedding tasks on $\mathcal{G}$. Previous approaches that construct a new graph from scratch typically require at least $O(n)$ variables to be determined [91, 77], which is computationally expensive. Contrarily, we adopt an intuitive yet effective weighted aggregation from the Laplacian matrices of all views into the MVAG Laplacian $\mathcal{L}$, where only r variables need to be optimized. In what follows, we describe the problem statement of multi-view attributed graph integration, while the objective function to solve the problem is formally developed in Section 5.3.

View Laplacians. Let $\mathbf{L}_i$ denote the Laplacian of the i-th view of $\mathcal{G}$, called the i-th view Laplacian. If this view is a graph view G_i, $\mathbf{L}_i$ is its normalized Laplacian $\mathbf{L}(G_i)$. If it is an attributed view $\mathbf{X}_i$, we adopt a prevalent way [82] to construct a K-nearest neighbor (KNN) graph $G_K(\mathbf{X}_i)$, consequently deriving its Laplacian $\mathbf{L}_i = \mathbf{L}(G_K(\mathbf{X}_i))$. In this KNN graph, every node is connected to K neighbors with the highest attribute similarity measured by cosine similarity, and each edge is weighted by attribute similarity.

MVAG Laplacian. Intuitively, each view in $\mathcal{G}$ can complement each other with its own information, and thus effective integration is vital to MVAG clustering and embedding. We define the MVAG Laplacian $\mathcal{L}$ as a weighted aggregation of all view Laplacians, where w_i is the i-th view weight.

$$\mathcal{L} = \sum_{i=1}^{r} w_i \mathbf{L}_i, \text{ where } \sum_{i=1}^{r} w_i = 1 \text{ and any } w_i \geq 0. \tag{5.1}$$

As a weighted combination of graph structures and attribute similarities across all views, this $\mathcal{L}$ can be interpreted as one integrated view of the MVAG and used for downstream tasks. For MVAG clustering, we employ the spectral clustering method

Table 5.1: Frequently used notations.

$\mathcal{G} = \{\mathcal{V}, \mathcal{E}_1, \ldots, \mathcal{E}_p, \mathbf{X}_{p+1}, \ldots, \mathbf{X}_r\}$	$\mathcal{G}$ is an MVAG with node set $\mathcal{V}$ and $r = p + q$ views, including p graph views with edge sets $\mathcal{E}_1, \ldots, \mathcal{E}_p$, and q attribute views $\mathbf{X}_{p+1}, \ldots, \mathbf{X}_r$.
n	The number of nodes in $\mathcal{G}$.
m_i, m	The number of edges in the i-th graph view, and the total number of edges in all graph views.
k	The number of clusters or classes in $\mathcal{G}$.
$G_K(\mathbf{X}_j)$	The K-nearest neighbor graph constructed from the attribute view $\mathbf{X}_j$.
$\mathbf{L}(G)$	The normalized Laplacian of a simple graph G.
$\mathbf{L}_i$	The i-th view Laplacian of $\mathcal{G}$.
$\mathcal{L}$	The MVAG Laplacian of $\mathcal{G}$, defined in (5.1).
$\mathbf{w} = [w_1, \ldots, w_r]$	A weight vector for r view Laplacians.
λ_i	The i-th smallest eigenvalue of $\mathcal{L}$.
$g_k(\mathcal{L})$	The eigengap objective.
$\lambda_2(\mathcal{L})$	The connectivity objective.
$h(\mathbf{w})$	The spectrum-guided objective function.
$\mathbf{w}^*$	The weights minimizing h found by `SGLA`.
$h_{\mathbf{\Theta}}(\mathbf{w})$, $h_{\mathbf{\Theta}^*}(\mathbf{w})$	An interpolation of h with coefficients $\mathbf{\Theta}$, or the optimal coefficients $\mathbf{\Theta}^*$.
$\mathbf{w}_0, \ldots, \mathbf{w}_r$	$r + 1$ weight vectors sampled for interpolation.
$\mathbf{w}^\dagger$	The weights minimizing $h_{\mathbf{\Theta}^*}$ found by `SGLA+`.

in [148] using the bottom eigenvectors of $\mathcal{L}$ to assign clusters. We utilize $\mathcal{L}$ as the input for graph embedding methods based on matrix factorization [99, 131] to enable them for MVAG embedding.

Problem Statement. The quality of $\mathcal{L}$ is crucial for empowering classic spectral clustering and network embedding methods to outperform state-of-the-art dedicated methods, while $\mathcal{L}$ solely depends on the r view weights in (5.1). Thus, our main research problem is to decide on proper view weights to produce an MVAG Laplacian for effective clustering and embedding quality.

5.3 SGLA Objective

Objective Overview. To assign the view weights for MVAG integration, trivial solutions such as utilizing a single view or allocating weights uniformly both compromise the performance, as validated in our experiments. Intuitively, we should cohesively leverage all views in the MVAG to produce $\mathcal{L}$, recognizing that different views may contribute variably. Community structures and connectivity properties are fundamental characteristics in real-world network data [26, 88], which are important for various problems, including clustering and embedding. To preserve these underlying properties, we analyze and design two objectives–*eigengap* and *connectivity*–which are combined in the full objective to produce the desired MVAG Laplacian $\mathcal{L}$.

In Section 5.3.1, we analyze and align the spectrum of $\mathcal{L}$ with the community property measured by normalized cut and propose an *eigengap objective*. In Section 5.3.2, we link the connectivity property, measured by conductance, to the spectrum of $\mathcal{L}$ and devise a *connectivity objective*. In Section 5.3.3, we combine the two objectives with an auxiliary regularization term to get the overall objective. The objective is formulated as a constrained nonlinear optimization, to find the desired view weights to compute $\mathcal{L}$. Figure 5.2 is a running example to intuitively explain the objectives in these sections.

5.3.1 Eigengap Objective

In this section, we aim to build the connection between the eigenvalues of the proposed MVAG Laplacian $\mathcal{L}$ and clustering quality that is measured by normalized cut in Definition 3. The normalized cut $\phi(\mathcal{C})$ of a cluster $\mathcal{C}$ in a simple graph G is the total weight of all outgoing edges from nodes within $\mathcal{C}$ to nodes outside $\mathcal{C}$ divided by the sum of degrees of all nodes in $\mathcal{C}$. A small normalized cut $\phi(\mathcal{C})$ indicates better cluster quality.

Definition 3. *In a graph G, a cluster of nodes $\mathcal{C} \subset \mathcal{V}$ has volume* $\mathrm{vol}(\mathcal{C}) = \sum_{v_a \in \mathcal{C}} \delta(v_a)$, *and* $\mathrm{Cut}(\mathcal{C}) = \sum_{v_a \in \mathcal{C}, v_b \notin \mathcal{C}} \mathbf{A}[a, b]$ *is its cut value that measures the total weight of outgoing edges from nodes within $\mathcal{C}$. The normalized cut of $\mathcal{C}$ is defined as* $\phi(\mathcal{C}) = \frac{\mathrm{Cut}(\mathcal{C})}{\mathrm{vol}(\mathcal{C})}$.

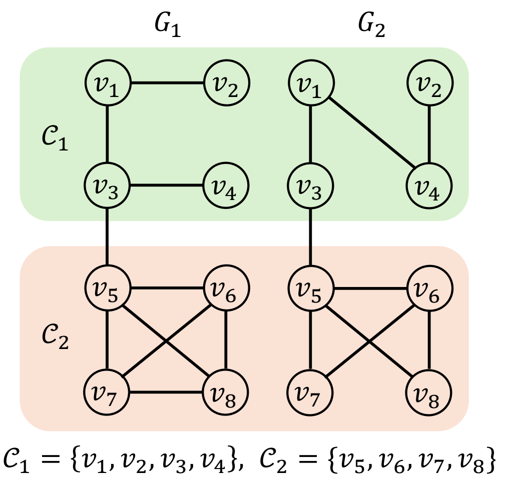
The high-level intuition is that if a simple graph G is perfectly clustered into k connected components, its normalized Laplacian forms a block diagonal matrix with zero-valued eigenvalues of multiplicity k, *i.e.*, $0 = \lambda_1 = \cdots = \lambda_k < \lambda_{k+1}$. According to matrix perturbation theory [56], a small perturbation of this Laplacian should keep $\lambda_1, \ldots, \lambda_k$ close to zero. Consequently, a graph with well-formed k clusters, *i.e.*, communities, should have small, near-zero eigenvalues $\lambda_1, ..., \lambda_k$, while eigenvalue λ_{k+1} is relatively larger, indicating *a significant multiplicative eigenvalue gap* between λ_{k+1} and λ_k. The following higher-order Cheeger's inequality from [66] provides an upper limit for $\phi(\mathcal{C})$. Using Theorem 5.3.1, we can establish a link between the eigenvalue gap and high-quality clusters, as shown in Corollary 5.3.1.1, by setting $\xi = \frac{1}{k}$.

Theorem 5.3.1. *There is a constant $c > 0$ such that for any weighted graph G and $k \in \mathbb{N}$, the following holds. Let $\xi \in (0, \frac{1}{3})$ be such that ξk is an integer. If $\lambda_{(1+\xi)k} > c\frac{(\log k)^2}{\xi^9}\lambda_k$, there are at least $s \geq (1 - 3\xi)k$ nonempty disjoint sets of nodes $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_s \subset \mathcal{V}$ such that $\phi(\mathcal{C}_i) \leq O(\sqrt{\frac{\lambda_k}{\xi^3}})$, $\forall 1 \leq i \leq s$.*

Corollary 5.3.1.1. *There is a constant $c > 0$ such that for any weighted graph G and $k \in \mathbb{N}$, the following holds. If $\lambda_k < \frac{1}{ck^9(\log k)^2}\lambda_{k+1}$, there are at least $s \geq k - 3$ nonempty disjoint clusters $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_s \subset \mathcal{V}$ such that the normalized cut $\phi(\mathcal{C}_i) \leq O(\sqrt{k^3\lambda_k})$, $\forall 1 \leq i \leq s$.*

Corollary 5.3.1.1 indicates that if the *multiplicative eigenvalue gap* between λ_k and λ_{k+1} is large enough, the normalized cut $\phi(\mathcal{C}_i)$ is bounded by $O(\sqrt{k^3\lambda_k})$ for some constant. This indicates an asymptotic upper bound for the normalized cut of clusters, associated with eigenvalues λ_k and λ_{k+1} of $\mathbf{L}(G)$.

Recall that the matrix $\mathcal{L}$ aggregated by the weighted sum of Laplacian matrices $\mathbf{L}_i$ over all r views in $\mathcal{G}$ in (5.1) should preserve the community information of all nodes

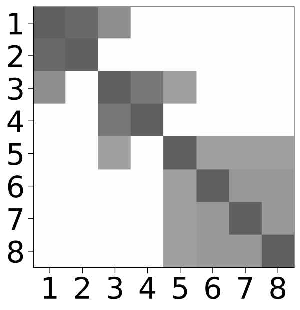


G_1
G_2
$\mathcal{C}_1$
$\mathcal{C}_2$
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8
$\mathcal{C}_1 = \{v_1, v_2, v_3, v_4\}$, $\mathcal{C}_2 = \{v_5, v_6, v_7, v_8\}$

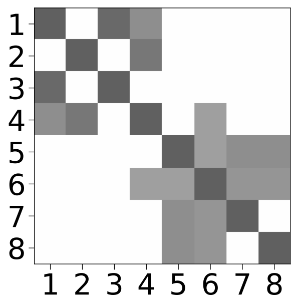
(a) Graph views G_1 and G_2 with 8 nodes in 2 clusters $\mathcal{C}_1$ and $\mathcal{C}_2$.

w_1	w_2	$g_k(\mathcal{L})$	$\lambda_2(\mathcal{L})$	$g_k - \lambda_2$
1.0	0.0	0.280	0.148	0.132
0.9	0.1	0.242	0.158	0.084
0.8	0.2	0.213	0.166	0.047
0.7	0.3	0.191	0.171	0.020
0.6	0.4	0.178	0.174	0.004
0.5	0.5	0.186	0.176	0.010
0.4	0.6	0.202	0.176	0.026
0.3	0.7	0.221	0.174	0.047
0.2	0.8	0.243	0.171	0.072
0.1	0.9	0.269	0.166	0.103
0.0	1.0	0.299	0.159	0.140

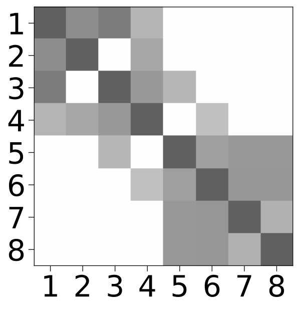
(b) Objective values with varied view weights w_1, w_2.



(c) $\mathbf{L}_1$ $(w_1 = 1, w_2 = 0)$.



(d) $\mathbf{L}_2$ $(w_1 = 0, w_2 = 1)$.



(e) $\mathcal{L}$ $(w_1 = 0.6, w_2 = 0.4)$.

Figure 5.2: A running example.

in $\mathcal{V}$. Suppose that the nodes in $\mathcal{G}$ are in k clusters. To preserve the well-formed community structures with low normalized cut bounded by Corollary 5.3.1.1, the aggregated matrix $\mathcal{L}$ should have as small $\frac{\lambda_k(\mathcal{L})}{\lambda_{k+1}(\mathcal{L})}$ as possible.

Consequently, our *eigengap objective* is to find desirable view weights w_i to obtain an $\mathcal{L}$ that *minimizes* the eigengap function $g_k(\mathcal{L})$ in (5.2) based on the spectrum of $\mathcal{L}$.

$$g_k(\mathcal{L}) = \frac{\lambda_k(\mathcal{L})}{\lambda_{k+1}(\mathcal{L})} \tag{5.2}$$

Example 1. *We use the two graph views G_1, G_2 shown in Figure 5.2a as an MVAG example for illustrating the eigengap objective. $\mathcal{G}$ contains 8 nodes in two ground-*

truth clusters $\mathcal{C}_1 = \{v_1, v_2, v_3, v_4\}$ and $\mathcal{C}_2 = \{v_5, v_6, v_7, v_8\}$, illustrated by different colors. Observe that, when only considering a single graph view, either G_1 or G_2, $\mathcal{C}_1$ does not exhibit a clear cluster structure due to the sparse connections in it per graph view, while the structure of $\mathcal{C}_2$ is clear in a single view. The observation is confirmed by the visualization of Laplacian matrices $\mathbf{L}_1$ and $\mathbf{L}_2$ of G_1 and G_2 in Figure 5.2c-5.2d, where values of larger magnitude are darker: the block formed by nodes v_1-v_4 in $\mathcal{C}_1$ are not cohesive, while the values for nodes v_5-v_8 illustrate a clear cluster for $\mathcal{C}_2$. The second column of Table 5.2b shows the eigengap values $g_k(\mathcal{L})$ when varying view weights w_1 and w_2 for aggregating the two view Laplacians. When assigning a large weight to a single view, the eigengap objective of the constructed $\mathcal{L}$ is large (e.g. 0.242 when $w_1 = 0.9, w_2 = 0$ or 0.269 when $w_1 = 0.1, w_2 = 0.9$), indicating that the clusters are not well-preserved by $\mathcal{L}$, due to the observations made above in Figure 5.2a, 5.2c and 5.2d. The eigengap objective is reduced to 0.178 when $w_1 = 0.6, w_2 = 0.4$, and we visualize the corresponding $\mathcal{L}$ in Figure 5.2e, in which, both clusters C_1 and C_2 can be clearly observed. Besides, the eigengap obtained by equal weights is 0.186, larger than 0.178. The example shows the intuition for minimizing the eigengap $g_k(\mathcal{L})$ to obtain proper view weights.

5.3.2 Connectivity Objective

The inherent connectivity of graphs is fundamental to the efficacy of graph algorithms [88]. Nevertheless, some graph views contain connection bottlenecks or leave certain nodes unconnected. Therefore, the MVAG Laplacian matrix $\mathcal{L}$ in (5.1) should amalgamate the connectivity of all views in $\mathcal{G}$. To this end, we formulate a connectivity objective that exploits the association between conductance and the spectrum of $\mathcal{L}$.

Given a simple graph G, its conductance $\Phi(G)$ measures how fast a random walk on G converges to its stationary distribution. In (5.3), $\Phi(G)$ is defined by the minimum normalized cut of the smaller side in all possible partitions.

$$\Phi(G) = \min_{\mathcal{C}|\mathrm{vol}(\mathcal{C})\leq \mathrm{vol}(\mathcal{V})/2} \frac{\mathrm{Cut}(\mathcal{C})}{\mathrm{vol}(\mathcal{C})}. \tag{5.3}$$

Higher graph conductance indicates stronger connectivity. However, computing the exact conductance is intractable in practice. From spectral graph theory [111], the graph conductance is bounded with λ_2, the second smallest eigenvalue of the normalized graph Laplacian.

$$\frac{\lambda_2}{2} \leq \Phi(G) \leq \sqrt{2\lambda_2}. \tag{5.4}$$

Obviously, a substantial λ_2 guarantees a lower bound for conductance. Therefore, we propose the *connectivity objective* to *maximize* the second smallest eigenvalue of $\mathcal{L}$, denoted by $\lambda_2(\mathcal{L})$, to preserve the overall connectivity in $\mathcal{G}$.

Example 2. *Recall that, in Figure 5.2a, cluster $\mathcal{C}_1$ has weak connectivity inside itself in both graph views G_1 and G_2 of $\mathcal{G}$. The third column in Table 5.2b shows the connectivity objective values $\lambda_2(\mathcal{L})$ of the obtained $\mathcal{L}$ when varying view weights. When a single view has a large weight, $\lambda_2(\mathcal{L})$ is smaller, indicating weak connectivity, which matches the observation above. If $\lambda_2(\mathcal{L})$ is sufficiently large, e.g., 0.174 when $w_1 = 0.6, w_2 = 0.4$, the obtained $\mathcal{L}$ can preserve the connectivity information from both views in $\mathcal{G}$, as illustrated in the corresponding visualization in Figure 5.2e, where the values for v_1, v_2, v_3, v_4 are clear to represent cluster $\mathcal{C}_1$.*

5.3.3 The Full Objective

In (5.5), we combine the eigengap objective $g_k(\mathcal{L})$ and connectivity objective $\lambda_2(\mathcal{L})$ into the full objective function $h(w_1, ..., w_r)$, also denoted by $h(\mathbf{w})$ where weight vector $\mathbf{w} = [w_1, \ldots, w_r]$. Since the view weights should minimize $g_k(\mathcal{L})$ while maximizing $\lambda_2(\mathcal{L})$, in (5.5), $\lambda_2(\mathcal{L})$ takes a negative sign, so that $h(\mathbf{w})$ is to be minimized. To prevent $\mathcal{L}$ from being dominated by a single view, we introduce a regularization term of all weights with parameter γ.

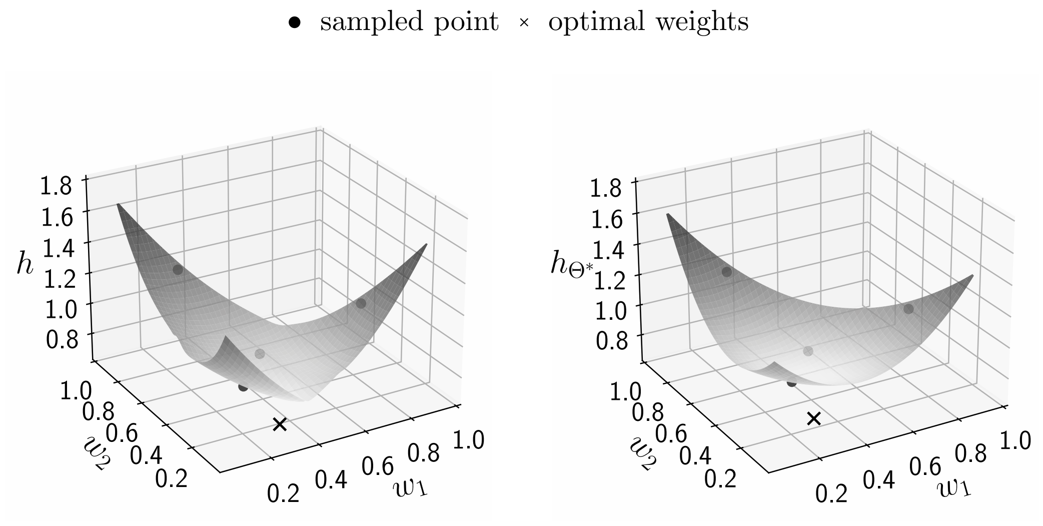
$$h(\mathbf{w}) = h(w_1, \ldots, w_r) = g_k(\mathcal{L}) - \lambda_2(\mathcal{L}) + \gamma \sum_{i=1}^{r} w_i^2 \tag{5.5}$$

The objective function h estimates the suitability of $\mathcal{L}$ for performing MVAG analytics and guides the search for appropriate view weights. Therefore, our problem statement in Section 5.2.2 is formulated by a constrained optimization problem, aiming to find the optimal weights $\mathbf{w}^* \in \mathbb{R}^r$ that minimize $h(\mathbf{w})$ while satisfying the constraints.

$$\begin{aligned} &\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} h(\mathbf{w}) = \operatorname{argmin}_{w_1,\ldots,w_r} h(w_1, \ldots, w_r), \\ &\text{s.t. any } w_i \geq 0 \text{ and } \sum_{i=1}^{r} w_i = 1. \end{aligned} \tag{5.6}$$

Discussion. Although both eigengap and connectivity objectives have associations with the measure of normalized cut, they each focus on different aspects of MVAGs. The eigengap objective is linked with the existence of multiple well-formed clusters, while the connectivity objective prevents nodes from being isolated from the main graph structure. Our combination of the two objectives aim to preserve both properties, achieving a balance between them. In the experiments, our method with the full objective in (5.6), combining both objectives, consistently outperforms approaches using a single objective.

Example 3. *To visualize the distribution of the objective $h(\mathbf{w})$ over all possible view weights, a case study is performed on Yelp dataset with three views (see Table 5.2 for details). Among the three view weights w_1, w_2, w_3, we vary w_1 and w_2 at interval* 0.01 *and set $w_3 = 1 - w_1 - w_2$, to exhaust all possible weight combinations, and we plot the value of $h(w_1, w_2, w_3)$ in Figure 5.3a. The plot shows a generally smooth surface that curves downward, which visually demonstrates the suitability of the proposed formulation.*



(a) Objective function h (b) Quadratic interpolation $h_{\mathbf{\Theta}^*}$

Figure 5.3: Plot of objective functions on Yelp.

5.4 Algorithms

In Section 5.4.1, we develop the `SGLA` method, which optimizes the objective to determine view weights. `SGLA` has demonstrated superior performance compared to existing methods. To further enhance efficiency while preserving result quality, we present `SGLA`+ in Section 5.4.2.

5.4.1 `SGLA` Method

Given an MVAG with r view Laplacians, the search space for possible view weights is exponential, rendering an exhaustive grid search intractable for solving (5.6). Optimizing the non-convex objective function $h(\mathbf{w})$ is further complicated by the presence of both inequality and equality constraints. Additionally, evaluating $h(\mathbf{w})$ and its gradients is costly due to intensive eigenvalue computations. Traditional gradient-descent methods are inefficient, as they require many iterations to converge and incur significant computational overhead.

To address the technical challenges, we develop the base method SGLA that produce high-quality results in a reasonable amount of time. SGLA iteratively performs two key computations: (i) *objective evaluation* and (ii) *objective optimization to update view weights*. Figure 5.4a provides an illustration of SGLA. After obtaining the Laplacian matrices $\mathbf{L}_1, ..., \mathbf{L}_r$ of the r graph views and attribute views in the input $\mathcal{G}$ as explained in Section 5.2.2 and initializing $\mathbf{w}$ with uniform weights, During objective evaluation, SGLA computes the latest $\mathcal{L}$ and the eigenvalues of $\mathcal{L}$ first and evaluates the objective function $h(\mathbf{w})$. Then, to update weights, SGLA optimizes and updates $\mathbf{w}$ via a derivative-free optimizer. This optimizer guarantees a local optimum when the variables in $\mathbf{w}$ converge. SGLA terminates either when the update of view weights is negligible, *i.e.*, convergence, or when the number of iterations exceeds a limit. SGLA eventually returns $\mathcal{L}$, which will be used for clustering and embedding tasks.

Algorithm. The pseudo code of SGLA is displayed in Algorithm 10. The weight parameters $w_1, ..., w_r$ of all r views are initialized to $\frac{1}{r}$ at Line 1. Then, from Lines 2 to 9, SGLA performs objective evaluation (Lines 3-5) and optimizes the proposed objective to update weights (Lines 6-9), in an iterative fashion for at most T_{max} iterations, and early terminates if the condition at Line 7 is met. Specifically, in an iteration, with the current weight parameters w_i, we first obtain the aggregated matrix $\mathcal{L}$ by weighted sum of the Laplacian matrices of all r views at Line 3. Then at Line 4, we compute the $k + 1$ smallest eigenvalues of $\mathcal{L}$. Note that all matrices, including $\mathbf{L}_i$ and $\mathcal{L}$, are organized as sparse matrices, and thus the computation at Lines 3 and 4 can be efficiently processed. With the eigenvalues λ_2, λ_k, and λ_{k+1}, we can compute the eigengap and connectivity objectives formulated in Section 5.3 to obtain $h(w_1, \ldots, w_r)$ by (5.5) (Line 5). Ω at Line 6 represents the constraints specified in (5.6). At Line 6, we adopt the optimizer Cobyla [97], considering the objective function and constraints, to update the first $r-1$ view weights to $w'_1, \ldots, w'_{r-1}$ as w'_r follows trivially. Briefly, this optimizer performs interpolation in the direction of each variable and updates a regular-shaped simplex of variables over iterations, as the trust

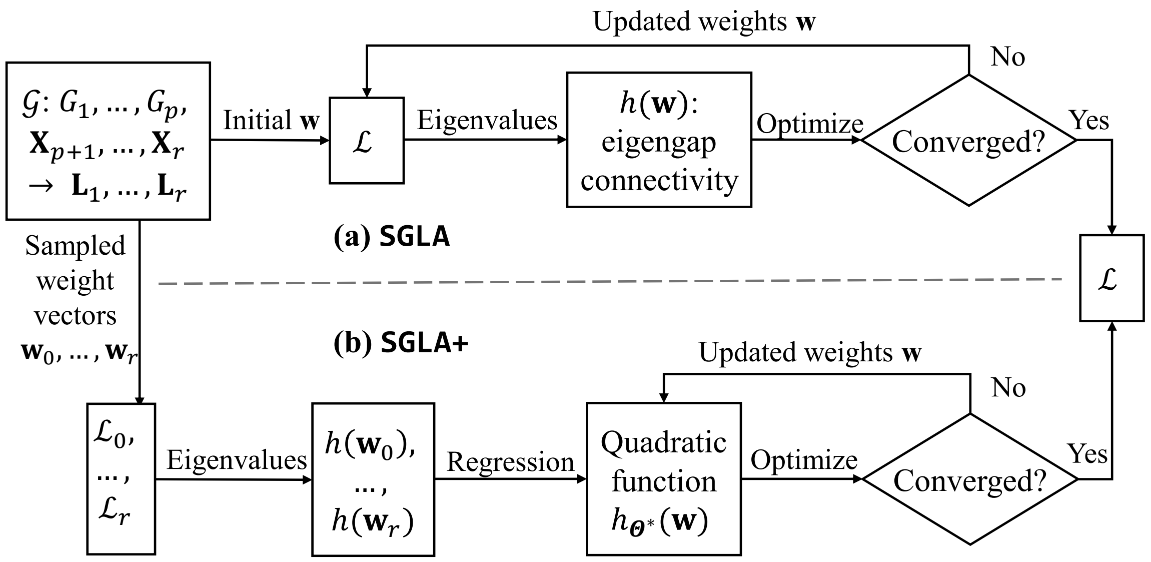

Updated weights w
No
$\mathcal{G}$: $G_1, \ldots, G_p$, $\mathbf{X}_{p+1}, \ldots, \mathbf{X}_r$ $\rightarrow$ $\mathbf{L}_1, \ldots, \mathbf{L}_r$
Initial w
$\mathcal{L}$
Eigenvalues
$h(\mathbf{w})$: eigengap connectivity
Optimize
Converged?
Yes
(a) SGLA
Sampled weight vectors $\mathbf{w}_0, \ldots, \mathbf{w}_r$
$\mathcal{L}$
(b) SGLA+
Updated weights w
No
$\mathcal{L}_0, \ldots, \mathcal{L}_r$
Eigenvalues
$h(\mathbf{w}_0), \ldots, h(\mathbf{w}_r)$
Regression
Quadratic function $h_{\boldsymbol{\Theta}^*}(\mathbf{w})$
Optimize
Converged?
Yes

Figure 5.4: Overview of SGLA and SGLA+ algorithms.

region that conforms to the constraints while minimizing the objective. The process stops when the updated variables are close to the previous iteration. At Line 7, if the difference between the updated view weights $w'_1, \ldots, w'_{r-1}$ and the previous weight values $w_1, \ldots, w_{r-1}$ measured by Euclidean distance, is below an early termination criteria ϵ, the iterative process of SGLA terminates and returns. Otherwise, we need to update the latest weights to $w_1, \ldots, w_{r-1}$ and w_r (Lines 8-9), and continue with the next iteration. Finally, the MVAG Laplacian matrix $\mathcal{L}$ is returned at Line 10, to be used in downstream tasks, including embedding and clustering.

Complexity. Given an MVAG $\mathcal{G}$ with n nodes and r views, all view Laplacian matrices are stored in sparse matrix format, and thus the costs of additions and matrix-vector multiplication operations on $\mathcal{L}$ are linear to the count of nonzero elements, *i.e.*, at most $2(m + qnK)$. The aggregation in Line 3 includes scalar multiplications and additions on Laplacian matrices. Line 4 solves eigenvalues via a bounded number of matrix-vector multiplications. Therefore, Lines 3-4 incur $O(m+qnK)$ time combined. Lines 5-6 and 8 are simple $O(r)$ computations, while the optimizer in Line 7 takes $O(r^2)$ time to update $r-1$ variables. r is typically small and can be regarded as a constant. When T iterations are conducted, we conclude that the overall time and

Algorithm 10: SGLA

```
Input: View Laplacians $\mathbf{L}_1, \ldots, \mathbf{L}_r$ of the input MVAG $\mathcal{G}$, number of clusters $k$,
       constraints $\Omega$, algorithm parameters $T_{\max}, \epsilon$.
1  Initialize weight parameters $w_1, \ldots, w_r \leftarrow \frac{1}{r}$ ;
2  for $t \leftarrow 1, \ldots, T_{max}$ do
3      $\mathcal{L} \leftarrow \sum_{i=1}^{r} w_i \mathbf{L}_i$;
4      $\lambda_1, \ldots, \lambda_{k+1} \leftarrow$ Eigenvalues$(\mathcal{L}, k+1)$;
5      Obtain $h(w_1, \ldots, w_r)$ by (5.5);
6      $w'_1, \ldots, w'_{r-1} \leftarrow$ Cobyla$(h(w_1, \ldots, w_r), \Omega)$;
7      if $\sqrt{\sum_{i=1}^{r-1} (w'_i - w_i)^2} < \epsilon$ then break;
8      $w_1, \ldots, w_{r-1} \leftarrow w'_1, \ldots, w'_{r-1}$;
9      $w_r \leftarrow 1 - \sum_{i=1}^{r-1} w_i$;
10 return $\mathcal{L}$;
```

space complexity of SGLA is $O(T(m + qnK))$.

5.4.2 SGLA+ Method

Although SGLA outperforms existing methods, it encounters significant computational challenges on large-scale MVAGs. The main bottleneck is the costly evaluation of the objective function $h(w_1, \ldots, w_r)$ (Lines 3-5 in Algorithm 10), which involves intensive eigenvalue computations. This evaluation must be repeated across many iterations for SGLA to converge, resulting in substantial overhead that limits its scalability.

To further improve efficiency, we design SGLA+ that has lower complexity than SGLA. SGLA+ has the following key designs. (i) Instead of directly optimizing the objective $h(\mathbf{w})$, SGLA+ formulates and optimizes an approximation $h_{\boldsymbol{\Theta}}$ of $h(\mathbf{w})$. This approximation is constructed as a quadratic interpolation that is quick to evaluate while closely resembling the original $h(\mathbf{w})$. (ii) We develop a sampling strategy to obtain $(r+1)$ weight vectors as samples of $h(\mathbf{w})$ to find an accurate approximation $h_{\boldsymbol{\Theta}^*}$, requiring only $(r+1)$ objective evaluations, fewer than those required by SGLA. (iii)

Finally, SGLA+ efficiently minimizes $h_{\mathbf{\Theta}^*}$ to compute the desired view weights necessary for constructing the MVAG Laplacian $\mathcal{L}$. Figure 5.4b provides an overview of SGLA+. Below, we first formulate the approximation of our objective, explain the sampling strategy, and then present the algorithm details.

Objective Approximation. The objective $h(w_1, \dots, w_r)$ in (5.6) has a weight vector $\mathbf{w} \in \mathbb{R}^r$ with r elements w_i, for $1 \leq i \leq r$, as variables. On the other hand, in optimization with a univariate objective, it is possible to find an approximate minimum by fitting a quadratic polynomial to three values of the objective [42]. Thus, we generalize quadratic interpolation to multiple variables. Specifically, given a set of weight vector samples, we aim to find a function $h_{\mathbf{\Theta}}(w_1, \dots, w_r)$ with a coefficient set $\mathbf{\Theta}$, as the interpolation of $h(w_1, \dots, w_r)$. As shown in (5.7), $h_{\mathbf{\Theta}}$ comprises all second-degree terms of w_i and w_j with coefficients $\theta_{i,j}$, where $1 \leq i \leq j \leq r-1$, linear terms w_i with coefficients $\theta_{i,r}$ for $i = 1, ..., r-1$, and a constant term $\theta_{r,r}$. We leave out w_r because it can be determined by the equality constraint that all weights sum up to 1. The matrix format of $h_{\mathbf{\Theta}}$ is in (5.8), where the upper triangular matrix $\mathbf{\Theta}$ consists of non-zero entries $\mathbf{\Theta}[i,j] = \theta_{i,j}$ for $i, j \in 1, ..., r$ and $i \leq j$.

$$h_{\mathbf{\Theta}}(\mathbf{w}) = \sum_{1 \leq i \leq j \leq r-1} \theta_{i,j} w_i w_j + \sum_{i=1}^{r-1} \theta_{i,r} w_i + \theta_{r,r} \tag{5.7}$$

$$h_{\mathbf{\Theta}}(\mathbf{w}) = \left[w_1, \dots, w_{r-1}, 1\right]^{\mathsf{T}} \mathbf{\Theta} \left[w_1, \dots, w_{r-1}, 1\right] \tag{5.8}$$

To determine the coefficients in $\mathbf{\Theta}$, we perform regression with $(r+1)$ weight vector samples $\mathbf{w}_\ell$, for $0 \leq \ell \leq r$. Sampling details will be explained shortly. Our goal is to find a solution $\mathbf{\Theta}^*$, such that the squared error between $h(\mathbf{w}_\ell)$ and $h_{\mathbf{\Theta}}(\mathbf{w}_\ell)$ over the $(r+1)$ samples is minimized in (5.9). We address this problem with a least Frobenius norm quadratic model [100]. Specifically, the optimization function in (5.9) exerts regularization $\|\mathbf{\Theta}\|_F^2$ with parameter α_r, to minimize the Frobenius norm of the coefficient matrix $\mathbf{\Theta}$, in order to find the desired solution.

$$\mathbf{\Theta}^* = \text{argmin}_{\mathbf{\Theta}} \left(\sum_{\ell=0}^{r} \big(h(\mathbf{w}_\ell) - h_{\mathbf{\Theta}}(\mathbf{w}_\ell)\big)^2 + \alpha_r \|\mathbf{\Theta}\|_F^2 \right) \tag{5.9}$$

After evaluating the objective function $h(\mathbf{w}_\ell)$ for each $\mathbf{w}_\ell$ among the $(r+1)$ weight vector samples, the regression in (5.9) can be solved via Cholesky decomposition. With coefficients $\mathbf{\Theta}^*$ found, the function $h_{\mathbf{\Theta}^*}$ is a local approximation of h. Subsequently, we find the minimum solution $\mathbf{w}^\dagger$ of $h_{\mathbf{\Theta}^*}$ under constraints, *i.e.*, (5.10), and $\mathbf{w}^\dagger$ is regarded as an approximate solution for the original problem in (5.6). This procedure is much more efficient than direct optimization of $h(\mathbf{w})$ in `SGLA`, as the evaluation of $h_{\mathbf{\Theta}^*}$ does not require the construction of $\mathcal{L}$ or the computation of eigenvalues.

$$\mathbf{w}^\dagger = \text{argmin}_{\mathbf{w}} h_{\mathbf{\Theta}^*}(\mathbf{w}) \text{ s.t. } w_i \geq 0 \;\forall\; 1 \leq i \leq r, \; \sum_{i=1}^{r} w_i = 1. \tag{5.10}$$

Weight Vector Sampling. The coefficient matrix $\mathbf{\Theta}$ contains $\frac{r(r+1)}{2}$ nonzero entries. Reaching a unique solution of $\mathbf{\Theta}$ requires $O(r^2)$ weight vector samples for expensive objective evaluations. Instead, we propose a scheme to gather $(r+1)$ weight vector samples only, which are sufficient in practice to find a high-quality $h_{\mathbf{\Theta}^*}$ via (5.9) and consequently $\mathbf{w}^\dagger$ in (5.10), as validated by experiments. The first sample $\mathbf{w}_0 = \left[\frac{1}{r}, \frac{1}{r}, \ldots, \frac{1}{r}\right] \in \mathbb{R}^r$ assigns the same weight $\frac{1}{r}$ for all r views of $\mathcal{G}$. Then for each ℓ-th view, a weight vector $\mathbf{w}_\ell$ is sampled as the midpoint between $\mathbf{w}_0$ and the one-hot vector $\mathbf{1}_\ell \in \{0,1\}^r$ that assigns a full weight to the ℓ-th view, where $1 \leq \ell \leq r$. Specifically, $\mathbf{w}_\ell = (\mathbf{w}_0 + \mathbf{1}_\ell)/2$, *i.e.*, the ℓ-th element in vector $\mathbf{w}_\ell$ has value $\frac{r+1}{2r}$ while the other elements have value $\frac{1}{2r}$.

Example 4. *For the Yelp dataset containing three views ($r = 3$), the objective function h is plotted in Figure 5.3a in Example 3. Following the sampling scheme explained above, we first obtain 4 weight vector samples $\mathbf{w}_0 = \left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right]$, $\mathbf{w}_1 = \left[\frac{2}{3}, \frac{1}{6}, \frac{1}{6}\right]$,*

Algorithm 11: `SGLA+`

```
Input: View Laplacians $\mathbf{L}_1, \ldots, \mathbf{L}_r$ of the input MVAG $\mathcal{G}$, number of clusters $k$,
       constraints $\Omega$, algorithm parameters $\alpha_r, T_{\max}, \epsilon$.
1  $\mathbf{w}_0 \leftarrow \left[\frac{1}{r}, \frac{1}{r}, \ldots, \frac{1}{r}\right] \in \mathbb{R}^r$;
2  for $\ell \leftarrow 0, \ldots, r$ do
3      if $\ell > 0$ then $\mathbf{w}_\ell \leftarrow (\mathbf{w}_0 + \mathbf{1}_\ell)/2$;
4      Obtain $\mathcal{L}_\ell$ by $\mathbf{w}_\ell$ and $\mathbf{L}_i$ via (5.1);
5      $\lambda_1, \ldots, \lambda_{k+1} \leftarrow$ Eigenvalues$(\mathcal{L}_\ell, k+1)$;
6      Obtain $h(\mathbf{w}_\ell)$ by (5.5);
7  Solve $\mathbf{\Theta}^*$ in (5.9) for observations $(\mathbf{w}_0, h(\mathbf{w}_0)), \ldots, (\mathbf{w}_r, h(\mathbf{w}_r))$ and L2 multiplier $\alpha_r$;
8  $w_1, \ldots, w_r \leftarrow \frac{1}{r}$ ;
9  for $t \leftarrow 1, \ldots, T_{max}$ do
10     Calculate $h_{\mathbf{\Theta}^*}(w_1, \ldots, w_r)$ by (5.8);
11     $w'_1, \ldots, w'_{r-1} \leftarrow$ Cobyla$(h_{\mathbf{\Theta}^*}(w_1, \ldots, w_r), \Omega)$, where $\Omega$ represents the constraints in
        (5.10);
12     if $\sqrt{\sum_{i=1}^{r-1} (w'_i - w_i)^2} < \epsilon$ then break;
13     $w_1, \ldots, w_{r-1} \leftarrow w'_1, \ldots, w'_{r-1}$;
14     $w_r \leftarrow 1 - \sum_{i=1}^{r-1} w_i$;
15 $\mathcal{L} \leftarrow \sum_{i=1}^{r} w_i \mathbf{L}_i$;
16 return $\mathcal{L}$;
```

$\mathbf{w}_2 = \left[\frac{1}{6}, \frac{2}{3}, \frac{1}{6}\right]$ *and* $\mathbf{w}_3 = \left[\frac{1}{6}, \frac{1}{6}, \frac{2}{3}\right]$. *Each of* $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ *emphasizes a specific view in the Yelp dataset. Figure 5.3a marks the locations of these four sampled points w.r.t. the weights of the first two views, as the third view weight is determined by the equality constraint that all weights add up to 1. Observe that the plot of* h *in Figure 5.3a resembles a partial paraboloid surface. According to (5.8), we can get* $h_{\mathbf{\Theta}}$ *on the Yelp dataset. The coefficients* $\mathbf{\Theta}^*$ *are determined by solving (5.9). We plot the acquired interpolation* $h_{\mathbf{\Theta}^*}$ *in Figure 5.3b, which exhibits a paraboloid surface similar to the original objective* h*. In addition, we use crosses in Figure 5.3a and 5.3b to mark the weights that minimize* h *and* $h_{\mathbf{\Theta}^*}$*, respectively. Their close locations validate that* $h_{\mathbf{\Theta}^*}$ *is an effective approximation for minimizing* h*.*

Algorithm. The pseudo code of `SGLA+` is provided in Algorithm 11. From Lines 1 to 7, we sample $(r+1)$ weight vectors, evaluate the objective $h(\mathbf{w}_\ell)$ over the sampled weight vectors, and solve $\mathbf{\Theta}^*$ to get the objective approximation $h_{\mathbf{\Theta}^*}$ at Line 7. From Lines 8 to 14, we aim to optimize $h_{\mathbf{\Theta}^*}$ by iteratively updating the view weights $w_1, \ldots, w_r$, and then the view weights are used to obtain $\mathcal{L}$ at Line 15. Specifically, the sampled weight vectors are first calculated, including the equal weights in $\mathbf{w}_0$ (Line 1), and then each $\mathbf{w}_\ell$ for $1 \leq \ell \leq r$ (Line 2-3). From each sampled vector $\mathbf{w}_\ell$, we construct the corresponding MVAG Laplacian $\mathcal{L}_\ell$ at Line 4. After solving its bottom $k+1$ eigenvalues (Line 5), the objective $h(\mathbf{w}_\ell)$ is evaluated at Line 6 as the observation for sample $\mathbf{w}_\ell$. At Line 7, interpolation over these $r+1$ samples is solved by regression with parameter α_r in (5.9) to obtain $\mathbf{\Theta}^*$. After initializing the target view weights $w_1, ..., w_r$ at Line 8, from Lines 9 to 14, we iteratively optimize the acquired interpolation $h_{\mathbf{\Theta}^*}$ to update $w_1, ..., w_r$. At each iteration (Line 9), $h_{\mathbf{\Theta}^*}(w_1, ..., w_r)$ is efficiently calculated at Line 10, without expensive construction of $\mathcal{L}$ or eigenvalue computation. Updating weights (Lines 11 to 14) and convergence condition (Line 12) are similar to `SGLA`, but are for the optimization of (5.10) instead of (5.6). After the termination of iterations, the converged weight parameters are used to construct $\mathcal{L}$ at Line 15.

Complexity. At Lines 1-6, `SGLA`+ performs exactly $(r+1)$ objective evaluations over all sampled weight vectors. The time and space complexity of this part is $O(r(m+qnK))$. To solve the $O(r^2)$ coefficients in $\mathbf{\Theta}$, Line 7 conducts a decomposition that theoretically takes $O(r^6)$ time. Since the number of views r is usually less than 10 and regarded as a constant, the cost of Line 7 can be considered negligible in practice. In Lines 9-14, Line 10 incurs $O(r^2)$ cost for evaluating $h_{\mathbf{\Theta}^*}$, and Lines 11-13 also have an $O(r^2)$ complexity combined. Hence, solving the weights over T iterations incurs negligible $O(Tr^2)$ cost. With r as constant, `SGLA`+ has an overall time and space complexity $O(m+qnK)$, as the processing on graph Laplacians is removed from the optimization loop. Observe that the complexity $O(m+qnK)$ of `SGLA`+ improves over the complexity $O(T(m+qnK))$ of `SGLA`. Empirically, we also find `SGLA`+ faster than

Table 5.2: Statistics of multi-view attributed graph datasets.

Dataset	n	r	m_i of G_i	d_j of $\mathbf{X}_j$	k
RM	91	11	267; 404; 298; 317; 163; 1,595; 1,683; 1,910; 1,565; 1,044	32	2
Yelp	2,614	3	262,859; 1,237,554	82	3
IMDB	3,550	3	5,119; 31,439	2,000	3
DBLP	4,057	4	3,528; 2,498,219; 3,386,139	334	4
Amazon photos	7,487	3	119,043	745; 7,487	8
Amazon computers	13,381	3	245,778	767; 13,381	10
MAG-eng	1,798,717	4	43,519,012; 10,112,848	1,000; 1,000	55
MAG-phy	2,353,996	4	257,706,767; 18,055,930	1,000; 1,000	22

`SGLA` in experiments.

5.5 Experiments

We evaluate the effectiveness and efficiency of our `SGLA` and `SGLA`+ over 12 competitors for clustering and 8 competitors for embedding on 8 real-world MVAG datasets. Section 5.5.1 describes experimental settings. Section 5.5.2 and 5.5.3 report the results of clustering and embedding, respectively. Section 5.5.4 conducts experimental analysis.

5.5.1 Experimental setup

Datasets. Table 5.2 provides the statistics of the 8 multi-view attributed graph datasets, including the number of nodes (n) and views (r), the number of edges m_i for each i-th graph view (separated by semicolons), dimension d_j for each j-th attribute view (separated by semicolons), and the number of ground truth node classes k that is also considered the number of clusters. These datasets are real-world MVAGs from diverse domains, including social activities (RM [6]), business (Yelp [81], Amazon photos and Amazon computers [91]), movies (IMDB [92]), and academic collaborations (DBLP [24], MAG-eng and MAG-phy [9]). The MAG-eng and MAG-phy datasets exemplify the complexity and the large scale of academic collaboration networks. MAG-phy includes over 2.35 million nodes with four views: two graph views with over 257.7 million and 18.05 million edges, and two high-dimensional attribute views with 1000 dimensions each. These datasets highlight the sparsity of graph views and the high dimensionality of attribute views, typical in real-world applications. Also, some datasets feature dense graph views, *e.g.*, DBLP, while others are sparse, *e.g.*, IMDB. The diversity, sparsity, and edge distributions of the datasets provide a representative testbed for comprehensive evaluation. The ground-truth clusters and class labels are obtained from the original data, *e.g.*, movie genres in IMDB and product categories in Amazon. In MAG-phy and MAG-eng, nodes are labeled by the subject domain of their publication venues.

Table 5.3: Clustering quality (the top 3 are in blue; darker shades indicate better results).

Method	**RM**					**Yelp**					**IMDB**					**DBLP**					**Overall rank**
	Acc	F1	NMI	ARI	Purity	Acc	F1	NMI	ARI	Purity	Acc	F1	NMI	ARI	Purity	Acc	F1	NMI	ARI	Purity	
WMSC	0.626	0.474	0.001	-0.017	0.703	0.813	0.836	0.537	0.489	0.813	0.374	0.291	0.003	0.001	0.379	0.780	0.778	0.468	0.507	0.780	9.7
2CMV	0.904	0.899	0.703	0.665	0.914	0.857	0.742	0.576	0.592	0.857	0.510	0.486	0.127	0.148	0.511	0.914	0.850	0.749	0.797	0.914	6.8
MEGA	0.802	0.793	0.423	0.359	0.802	0.653	0.568	0.390	0.427	0.733	0.390	0.239	0.007	0.004	0.392	0.913	0.907	0.741	0.792	0.913	8.3
HDMI	0.613	0.459	0.010	-0.018	0.703	0.909	0.915	0.681	0.727	0.909	0.541	0.547	0.162	0.142	0.532	0.895	0.885	0.706	0.761	0.896	8.5
URAMN	0.736	0.684	0.107	0.195	0.736	0.771	0.762	0.490	0.483	0.771	0.588	0.582	0.183	0.197	0.588	0.908	0.901	0.715	0.781	0.896	6.0
O2MAC	0.659	0.397	0.040	-0.044	0.703	0.649	0.565	0.391	0.425	0.732	0.547	0.550	0.135	0.139	0.535	0.873	0.865	0.669	0.705	0.877	9.5
DMG	0.745	0.623	0.147	0.191	0.765	0.714	0.725	0.441	0.365	0.714	0.545	0.459	0.195	0.209	0.550	0.925	0.921	0.761	0.815	0.925	6.3
LMGEC	0.703	0.500	0.015	0.044	0.703	0.923	0.928	0.725	0.764	0.923	0.568	0.577	0.166	0.143	0.562	0.922	0.917	0.757	0.813	0.922	5.1
MAGCN	0.703	0.736	0.000	0.000	0.703	0.734	0.705	0.437	0.455	0.734	0.513	0.482	0.116	0.135	0.511	-	-	-	-	-	9.5
MCGC	0.967	0.959	0.799	0.867	0.967	0.860	0.874	0.596	0.597	0.860	0.567	0.545	0.164	0.186	0.553	0.902	0.895	0.716	0.771	0.902	4.6
MvAGC	0.774	0.710	0.267	0.329	0.790	0.907	0.915	0.685	0.720	0.907	0.552	0.462	0.191	0.201	0.549	0.874	0.866	0.650	0.708	0.874	5.5
MAGC	0.714	0.451	0.040	0.030	0.714	0.564	0.520	0.413	0.315	0.565	0.484	0.424	0.057	0.062	0.485	0.928	0.923	0.771	0.827	0.928	7.4
SGLA	0.978	0.974	0.830	0.911	0.978	0.927	0.930	0.727	0.779	0.927	0.559	0.455	0.211	0.223	0.558	0.934	0.930	0.789	0.841	0.933	1.7
SGLA+	1.000	1.000	1.000	1.000	1.000	0.930	0.934	0.740	0.786	0.930	0.554	0.450	0.210	0.220	0.555	0.930	0.925	0.775	0.831	0.930	2.0

Method	**Amazon photos**					**Amazon computers**					**MAG-eng**					**MAG-phy**					**Overall rank**
	Acc	F1	NMI	ARI	Purity	Acc	F1	NMI	ARI	Purity	Acc	F1	NMI	ARI	Purity	Acc	F1	NMI	ARI	Purity	
WMSC	0.323	0.285	0.152	0.103	0.392	0.248	0.191	0.165	0.090	0.375	-	-	-	-	-	-	-	-	-	-	9.7
2CMV	0.523	0.434	0.450	0.332	0.638	0.309	0.269	0.312	0.135	0.524	-	-	-	-	-	-	-	-	-	-	6.8
MEGA	0.328	0.292	0.265	0.043	0.427	0.319	0.170	0.230	0.081	0.483	-	-	-	-	-	-	-	-	-	-	8.3
HDMI	0.273	0.126	0.026	0.018	0.289	0.303	0.111	0.034	0.027	0.375	-	-	-	-	-	-	-	-	-	-	8.5
URAMN	0.669	0.642	0.521	0.427	0.689	0.353	0.280	0.364	0.163	0.588	-	-	-	-	-	-	-	-	-	-	6.0
O2MAC	0.307	0.153	0.087	0.012	0.298	0.340	0.100	0.020	0.034	0.380	-	-	-	-	-	-	-	-	-	-	9.5
DMG	0.603	0.548	0.508	0.391	0.671	0.401	0.295	0.384	0.200	0.598	-	-	-	-	-	-	-	-	-	-	6.3
LMGEC	0.626	0.606	0.530	0.423	0.703	0.410	0.304	0.374	0.240	0.614	-	-	-	-	-	-	-	-	-	-	5.1
MAGCN	0.528	0.454	0.456	0.314	0.587	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9.5
MCGC	0.674	0.582	0.595	0.449	0.754	0.569	0.501	0.557	0.419	0.726	-	-	-	-	-	-	-	-	-	-	4.6
MvAGC	0.615	0.568	0.558	0.384	0.726	0.516	0.426	0.512	0.365	0.697	0.256	0.108	0.355	0.139	0.293	0.314	0.107	0.238	0.022	0.376	5.5
MAGC	0.646	0.571	0.591	0.384	0.687	0.447	0.438	0.323	0.158	0.481	-	-	-	-	-	-	-	-	-	-	7.4
SGLA	0.786	0.710	0.670	0.622	0.819	0.585	0.507	0.589	0.441	0.740	0.464	0.369	0.575	0.332	0.597	0.582	0.455	0.620	0.449	0.725	1.7
SGLA+	0.782	0.705	0.657	0.618	0.815	0.604	0.515	0.577	0.426	0.744	0.455	0.352	0.570	0.329	0.583	0.561	0.413	0.608	0.439	0.704	2.0

Baselines. For embedding, we compare with 8 baselines, including 3 attributed network embedding methods PANE [143], AnECI [79] and CONN [114] that are applied to a multi-view attributed graph by aggregating the graph adjacency matrices and concatenating the attribute views, 3 attributed multiplex graph embedding methods O2MAC [24], HDMI [49] and URAMN [152] that are applied to a multi-view attributed graph by concatenating the attribute views when necessary, and 2 multi-view attributed graph embedding methods DMG [86] and LMGEC [30]. For clustering, we compare with 12 baselines, including 8 multi-view attributed graph clustering approaches, namely WMSC [160], MEGA [123] adapted for unsupervised clustering, 2CMV [82], LMGEC, MAGCN [14], MCGC [91], MvAGC [76], and MAGC [77], and 4 embedding methods HDMI, URAMN, O2MAC, and DMG coupled with spectral clustering.

Implementation. Across all datasets, SGLA and SGLA+ adopt the same parameter settings $\gamma = 0.5$, $\epsilon = 0.001$, $T_{\max} = 50$, and $\alpha_r = 0.05$. We also conduct experiments to vary parameters. We set $K = 10$ for KNN graphs by default. For Yelp and IMDB, we use $K = 200$ and 500, respectively, since their attribute views are more informative. A larger K incorporates more attribute similarity connections in the KNN graphs. Source codes of all competitors are obtained from the respective authors, each tuned with parameters suggested in the respective paper. We fix the embedding dimension to 64. Experiments are conducted on a Linux computer with Intel Xeon 6226R CPU, RTX3090 GPU, and 384 GB RAM. A maximum of 16 CPU threads are available. Note that CONN, HDMI, URAMN, DMG, and LMGEC are GPU-powered, while the other methods, including our SGLA and SGLA+, run on CPU.

Evaluation Settings. The clustering quality is measured by accuracy (Acc), average per-class macro-F1 score (F1), normalized mutual information (NMI), adjusted Rand index (ARI), and Purity score with respect to ground truth. ARI ranges from -0.5 to 1, whereas the other 4 metrics are in range $[0, 1]$. The node embedding for classification is evaluated by Macro-F1 (MaF1) and Micro-F1 (MiF1). For all these metrics, a larger value indicates better performance. Efficiency is measured by the total running time in seconds. Results are averaged over 5 repeated runs. In Tables 5.3 and 5.4, a ‘ - ’ indicates the method cannot produce results within one day or

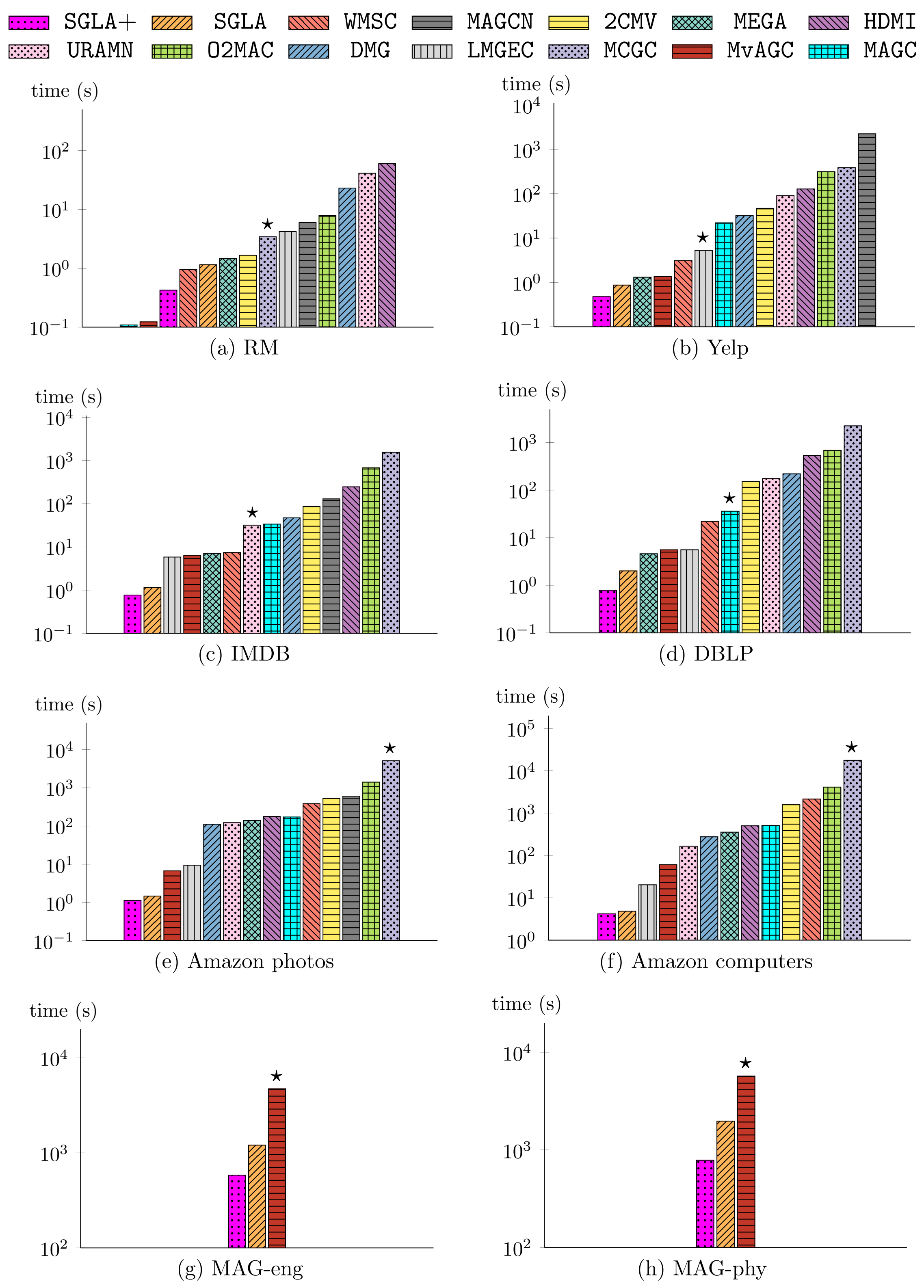


Figure 5.5: Running time of clustering in seconds (★ marks the competitor with the best clustering quality in Table 5.3).

Table 5.4: Embedding performance for node classification (the top 3 are in blue; darker shades indicate better results).

Method	RM		Yelp		IMDB		DBLP		Amazon photos		Amazon computers		MAG-eng		MAG-phy		Overall rank
	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	MaF1	MiF1	
PANE	0.738	0.778	0.904	0.902	0.479	0.494	0.636	0.763	0.783	0.847	0.556	0.674	0.550	0.672	0.547	0.674	6.0
AnECI	0.539	0.734	0.778	0.826	0.589	0.596	0.880	0.894	0.899	0.915	0.807	0.846	-	-	-	-	6.1
CONN	0.569	0.751	0.932	0.926	0.657	0.657	0.725	0.758	0.892	0.914	0.827	0.850	-	-	-	-	4.6
HDMI	0.446	0.666	0.926	0.918	0.641	0.642	0.916	0.922	0.724	0.792	0.500	0.721	-	-	-	-	5.7
URAMN	0.496	0.690	0.916	0.907	0.640	0.653	0.897	0.905	0.580	0.727	0.313	0.651	-	-	-	-	6.7
O2MAC	0.689	0.745	0.898	0.894	0.657	0.657	0.909	0.915	0.672	0.721	0.442	0.606	-	-	-	-	6.1
DMG	0.692	0.737	0.902	0.891	0.618	0.624	0.928	0.933	0.796	0.874	0.629	0.757	-	-	-	-	5.2
LMGEC	0.417	0.717	0.938	0.932	0.597	0.608	0.916	0.922	0.630	0.723	0.347	0.669	-	-	-	-	6.2
SGLA	0.835	0.904	0.941	0.936	0.688	0.687	0.934	0.938	0.918	0.933	0.893	0.907	0.574	0.736	0.702	0.830	1.5
SGLA+	0.856	0.918	0.942	0.937	0.705	0.704	0.932	0.937	0.912	0.929	0.880	0.901	0.588	0.741	0.696	0.827	1.5

runs out of memory.

5.5.2 Effectiveness and Efficiency on Clustering

SGLA and SGLA+ generate $\mathcal{L}$ which is then used as the input of spectral clustering as described in Section 5.2.2. In Table 5.3, we report the Acc, F1, NMI, ARI, Purity and the averaged overall rank of each method over all 8 datasets across the four metrics. We highlight the top-3 best results on each dataset in blue with darker shades indicating better performance.

Effectiveness. As shown in the last column of Table 5.3, SGLA and SGLA+ achieve the best ranks 1.7 and 2.0 respectively over all 8 datasets, significantly outperforming the best competitor with rank 4.6. Specifically, for the 5 metrics, SGLA and SGLA+ achieve better performance compared to existing methods on almost all datasets, except Acc, F1 and Purity on IMDB. For the RM, Yelp, IMDB, and DBLP datasets in Table 5.3, SGLA and SGLA+ achieve improvements in NMI over the best baseline by up to 20.1%, 1.5%, 1.6% and 1.8%, respectively. On Amazon photos, SGLA and SGLA+ outperform the best competitor MCGC by large margins up to 11.2% in Acc, 12.8% in F1, 7.5% in NMI, 17.3% in ARI and 6.5% in Purity. Remarkably, on the large-scale MAG-eng and MAG-phy datasets in Table 5.3, SGLA and SGLA+ surpass the only scalable baseline MvAGC by up to 23.8% in Acc, 30.5% in F1, 30.1% in NMI, 31.0% in ARI and 32.7% in Purity on average. The results in Table 5.3 validate the effectiveness of the objective formulated in Section 5.3 and the techniques in 5.4 to solve it, which aligns the spectrum of $\mathcal{L}$ with the community and connectivity properties. On the other hand, MCGC aligns a unified graph with each view but overlooks its intrinsic structure, leading to inferior performance. The results in Table 5.3 underscore that SGLA and SGLA+ effectively generate a MVAG Laplacian $\mathcal{L}$ that reliably reveals the underlying clusters in real-world multi-view attributed graphs.

Efficiency. For our methods, we record the *total time cost* of computing view Laplacians from $\mathcal{G}$, running SGLA or SGLA+, and performing clustering. Figure 5.5 displays the running time of all methods, with the competitor delivering the best clustering quality marked by a star for each dataset. The y-axis is time in seconds on a logarithmic scale. Regardless of clustering quality, SGLA and SGLA+ consistently demonstrate leading efficiency across all datasets except RM. Compared to the marked competitors, SGLA and SGLA+ are often faster by orders of magnitude. For instance, on DBLP, the best baseline MAGC requires 35.98 seconds to finish, while SGLA and SGLA+ only take 2.008 and 0.788 seconds, respectively, achieving a 17.9× and 45.7× speedup. On Amazon photos, the marked baseline MCGC requires 5102 seconds, whereas our methods SGLA and SGLA+ take only 1.465 and 1.129 seconds, attaining a significant speedup of over 3000×. On MAG-eng and MAG-phy, where most baselines run out of memory or cannot finish within one day, our methods achieve the highest efficiency with the best quality. Our methods demonstrate a huge speedup over baselines with quadratic complexity, *e.g.*, MAGC, and the GNN models that are expensive to train, *e.g.*, HDMI. Furthermore, SGLA+ is consistently faster than SGLA on all datasets. For example, SGLA requires 1206 and 1970 seconds for clustering MAG-eng and MAG-phy, respectively, while SGLA+ requires only 583 and 783 seconds.

Moreover, for memory usage, SGLA and SGLA+ also exhibit high memory efficiency over baselines. For instance, on large-scale datasets, our methods only use 18.7 GB for MAG-eng and 32.3 GB for MAG-phy, while MvAGC requires 137 GB and 184 GB, respectively, and all other baselines are out of memory. These results highlight the efficacy of the algorithm designs presented in Section 5.4.

5.5.3 Effectiveness and Efficiency on Embedding

SGLA and SGLA+ generate $\mathcal{L}$ which is then used as the input for classic network embedding methods as described in Section 5.2.2. Specifically, on large datasets MAG-eng and MAG-phy, the scalable SketchNE [131] is utilized, while NetMF [99] is used for the remaining datasets. We evaluate the embedding quality by node classification task. Specifically, for each method that outputs embeddings, a logistic regression classifier is trained on 20% of the ground truth class labels (1% for the large MAG-eng and MAG-phy datasets), with the remaining labels used for testing. In Table 5.4, we report the embedding quality of our methods and the baselines over all datasets, in terms of classification performance (MaF1, MiF1). Figure 5.6 compares the efficiency of all methods, measured by total embedding time in seconds.

Effectiveness. In Table 5.4, our methods SGLA and SGLA+ consistently claim the top two places, with the best overall rank 1.5 over all metrics on all datasets, significantly higher than the best competitor with overall rank 4.6. For example, on IMDB in Table 5.4, SGLA+ and SGLA take the first and second places respectively, surpassing O2MAC and CONN, which rank third, by up to 4.8% in Macro-F1 and 4.7% in Micro-F1. On the Amazon computers dataset, our methods outperform the runner-up CONN by up to 6.6% in Macro-F1 and 5.7% in Micro-F1. Moreover, on the large datasets, MAG-eng and MAG-phy, our methods SGLA+ and SGLA surpass PANE, the sole competitor with sufficient scalability, proving that our methods are more effective in producing high-quality embeddings for these large-scale datasets. Compared with sophisticated GNN methods focused on the common and specific aspects of each view, *e.g.*, DMG, our methods better preserve structural properties, thus allowing classic methods to produce high-quality node embeddings for MVAGs. The results in Table 5.4 confirm the effectiveness of the objective and techniques developed in Section 5.3 and 5.4.

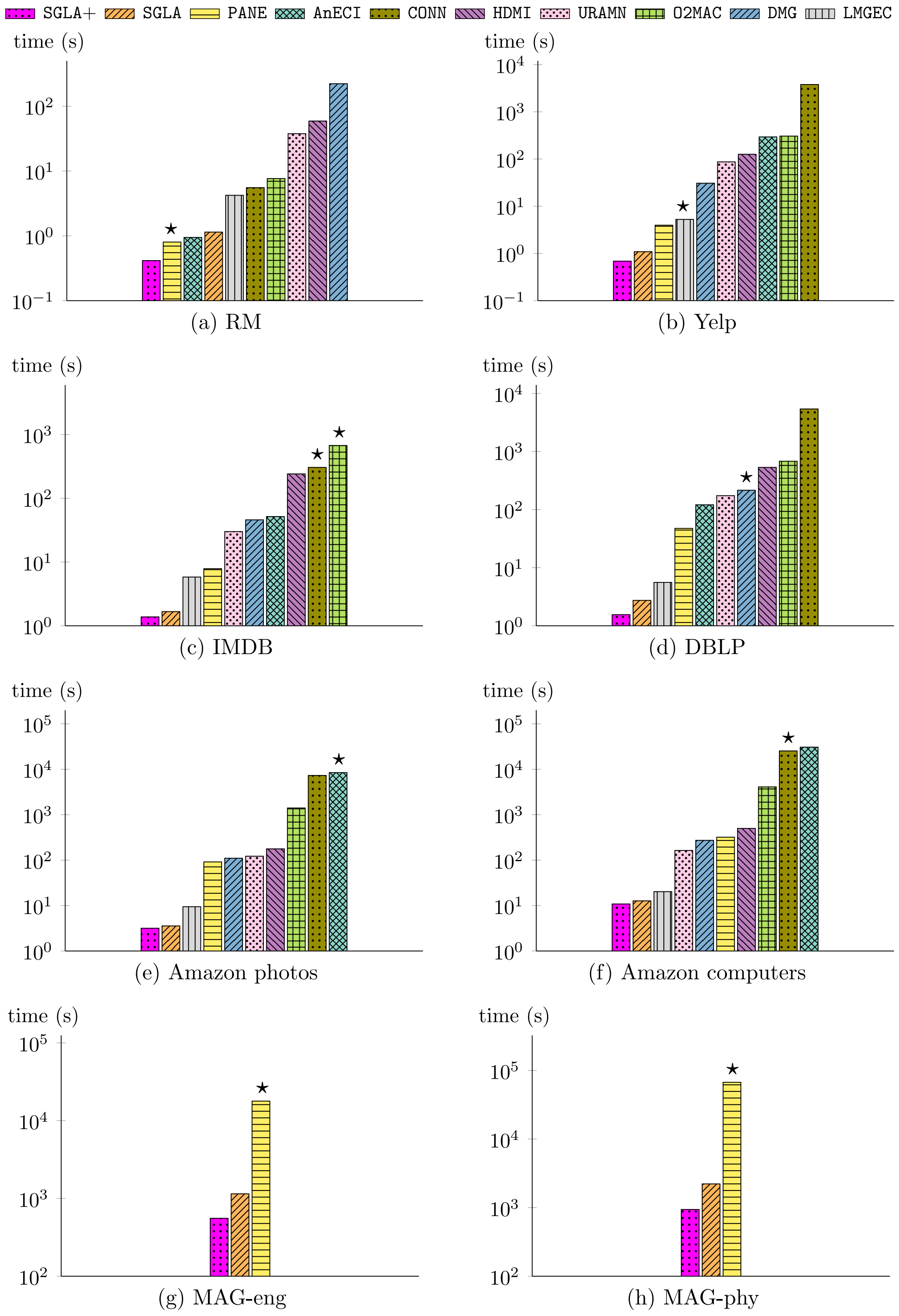


Figure 5.6: Running time of embedding in seconds (⋆ marks the competitor with the best embedding quality in Table 5.4).

Efficiency. Figure 5.6 displays the *total time cost* for MVAG embedding on 8 datasets, with the competitors delivering the best embedding quality marked for each dataset. The y-axis represents running time in seconds on a logarithmic scale. SGLA+ achieves the best efficiency on all datasets, often outperforming the compared methods by orders of magnitude; SGLA is also faster than all baselines except on the smallest dataset, RM. For instance, in Figure 5.6c, while CONN and O2MAC are the runner-ups after SGLA and SGLA+ in quality, our methods are up to 222× and 489× faster, respectively. On the million-scale datasets MAG-eng and MAG-phy, SGLA+ requires 555 and 939 seconds each, achieving 32.1× and 71.6× speedup over the only scalable baseline PANE. Moreover, SGLA+ is faster than SGLA on all datasets. For memory usage, our methods also achieve high space efficiency. For example, to produce embeddings for large-scale datasets MAG-eng and MAG-phy, both SGLA and SGLA+ leave a memory footprint of 61.6 GB and 95.4 GB, respectively. However, the only scalable baseline PANE requires 221 GB and 299 GB, while all other methods run out of memory. These results highlight the efficacy of the algorithm designs presented in Section 5.4.

5.5.4 Experimental Analysis

Convergence evaluation and $T_{\max}$. When varying the number of iterations t in SGLA, Figure 5.7 shows the convergence of the objective $h(\mathbf{w})$ that decreases and then becomes stable, while the corresponding clustering accuracy (Acc) improves. The black dots mark the iteration when the termination condition by ϵ at Line 7 of Algorithm 10 is met. Observe that $h(\mathbf{w})$ usually converges before termination. Thus, we set $T_{\max} = 50$ by default at Line 2 of Algorithm 10.

Varying ϵ. We vary parameter ϵ in the termination condition from 10^{-4} to 10^{-1} (from tight to loose) and report the results of SGLA in Figure 5.8. Compared to the default setting of $\epsilon = 10^{-3}$, Δtime denotes the ratio of change in running time. As

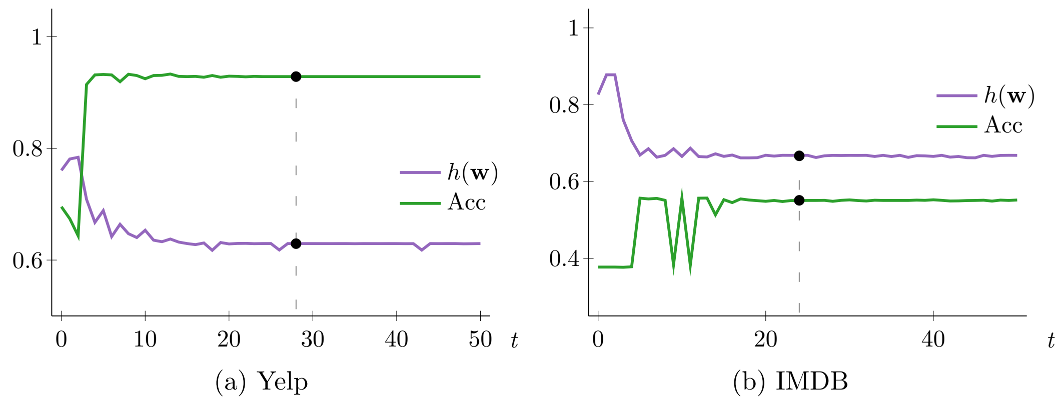


Figure 5.7: Varying number of iterations t in `SGLA` for clustering accuracy; ● marks when termination condition is met).

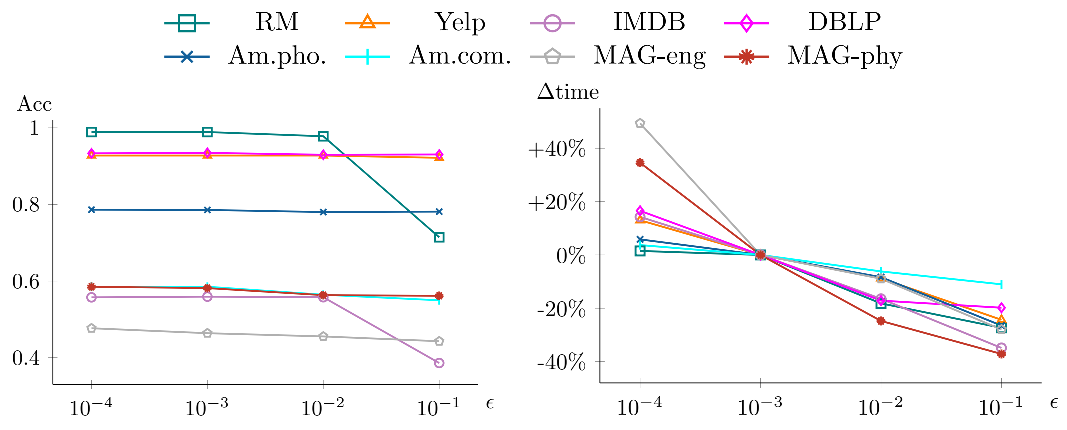


Figure 5.8: Varying ϵ for `SGLA`.

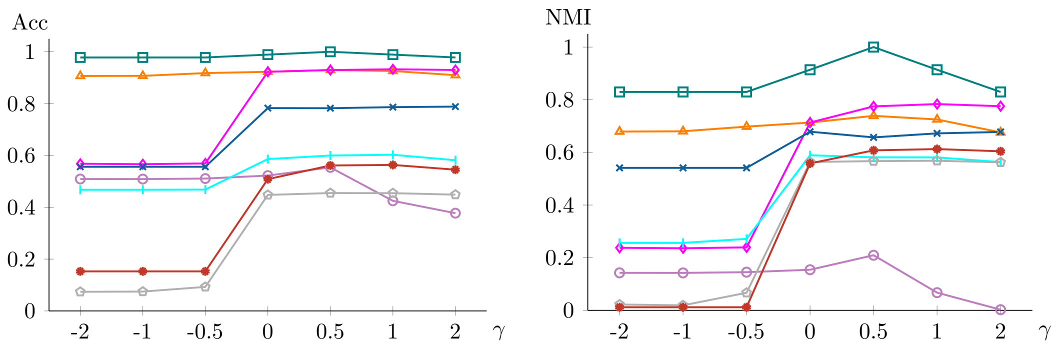


Figure 5.9: Varying γ for `SGLA+`.

ϵ becomes loose from 10^{-4} to 10^{-1}, the clustering quality (Acc) is stable first and then decreases. On the other hand, as ϵ becomes tight, e.g., from 10^{-3} to 10^{-4}, the

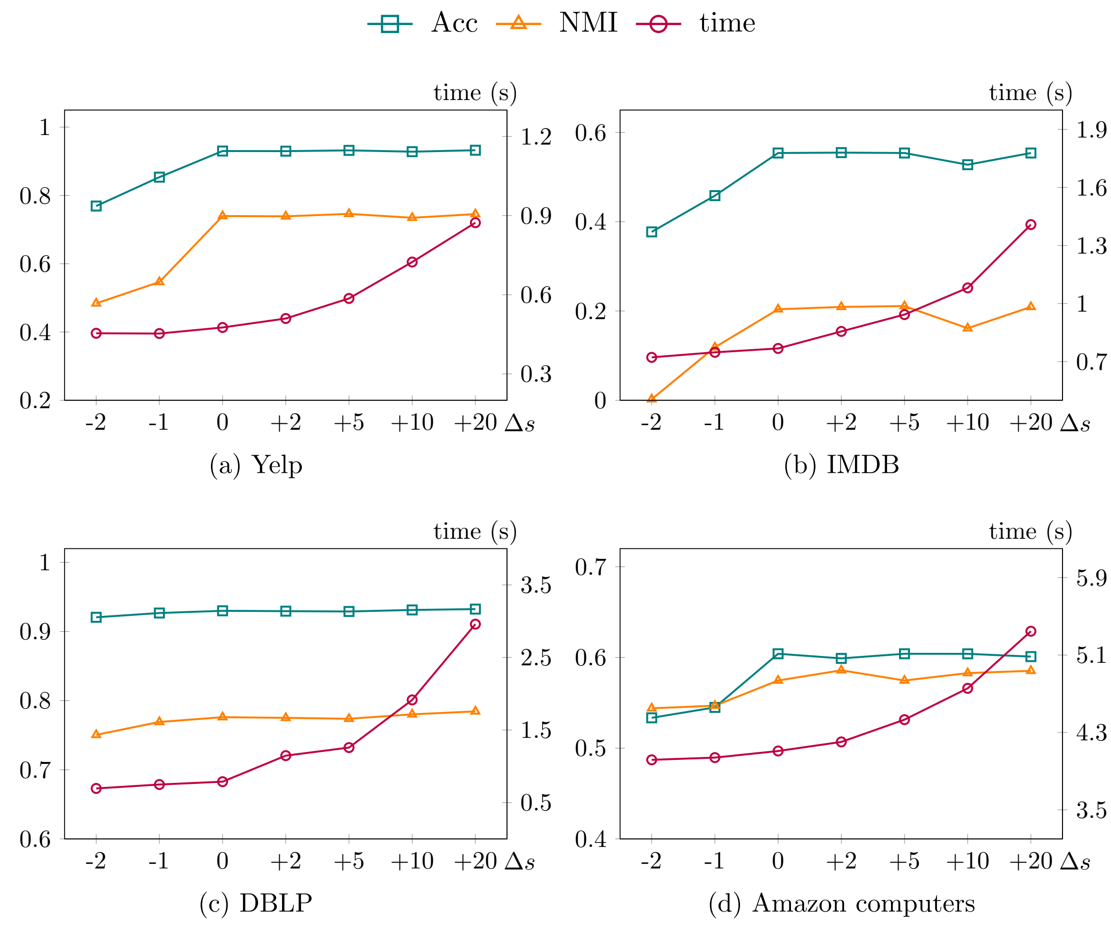


Figure 5.10: Vary the number of weight vector samples in `SGLA+`.

running time increases significantly (Δtime) but the quality (Acc) maintains. Thus, we set $\epsilon = 0.001$ by default.

Varying γ. Parameter γ is the coefficient of the regularization term in (5.5). A negative γ promotes weight allocation to focus on a single view, while a positive γ mitigates such situations and tends to assign similar weights across views. We vary γ from -2 to 2 and report the accuracy and NMI scores in Figure 5.9. As γ increases from -2 to 0.5, the accuracy and NMI of `SGLA+` remain relatively stable on Yelp and show noticeable improvement on other datasets. However, when γ varies from 0.5 to 2, the accuracy and NMI degrade on IMDB and RM, while remaining stable on other datasets. Based on these observations, we set $\gamma = 0.5$ by default.

Varying the number of weight vector samples in `SGLA+`. In Section 5.4.2, `SGLA+` uses $(r+1)$ sampled weight vectors by default. We vary the number of samples and change $(r+1)$ by $\Delta s \in \{-2, -1, 0, +2, +5, +10, +20\}$, and report the results of `SGLA+` in Figure 5.10, where the left y-axis is for clustering Acc and NMI, and the right y-axis is for running time. The removed (resp. added) samples are randomly selected (resp. generated). Observe that when the number of samples changes by Δs from -2 to 0, Acc and NMI scores increase and then become stable afterward with larger delta values. Meanwhile, the time increases significantly due to more expensive objective evaluations to be performed. The results in Figure 5.10 indicate that $(r+1)$ samples are sufficient in practice.

Alternative integrations. For the proposed spectrum-guided integration in `SGLA+` that optimizes the full objective, we compare baselines optimizing the connectivity or eigengap objective alone, setting equal weights for all view Laplacians (Equal-w), and directly aggregating adjacency matrices from graph views and KNN graphs of attribute views (Graph-Agg). Figure 5.11 reports the clustering accuracy on each dataset and the average accuracy over all datasets. `SGLA+` is the best in average accuracy performance and achieves the highest accuracy on almost all datasets. Optimizing connectivity or eigengap alone can achieve relatively good accuracy on some datasets but performs poorly on others, validating the design choice to combine both objectives. Despite occasional successes, assigning equal view weights often leads to poor performance, as evidenced by the low clustering quality on datasets such as RM, Yelp, and IMDB. Graph-Agg is outperformed by `SGLA+` that adopts normalized Laplacians and preserves intrinsic spectrum properties of MVAGs. These results highlight the advantage of our spectrum-guided multi-view integration.

Embedding visualization. We visualize the node embeddings using t-SNE [118] to qualitatively assess the embedding quality. Due to space constraints, we present the visualizations of our method `SGLA+` and the strong baselines identified in Table 5.4 on the RM and Yelp datasets in Figure 5.12. On RM, shown in the 1st row of Figure

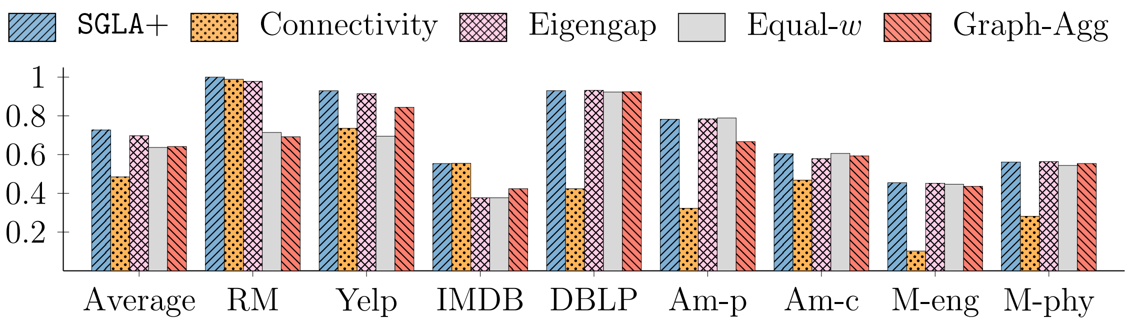


Figure 5.11: Clustering accuracy with alternative integrations.

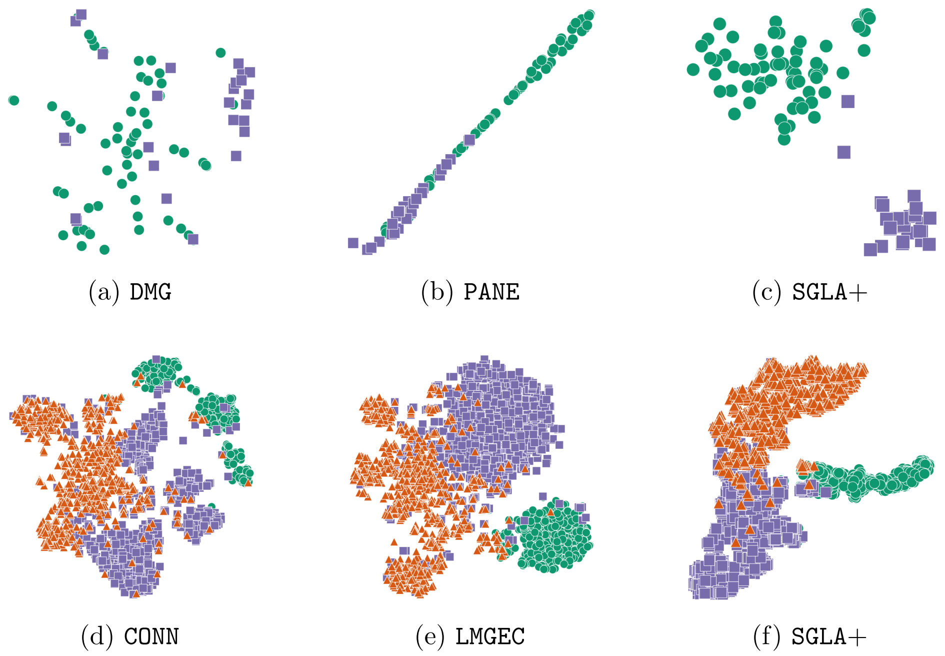


Figure 5.12: Embedding visualization on RM (a,b,c) and Yelp (d,e,f). Ground-truth classes are in color.

5.12, SGLA+ effectively separates nodes into different classes, while DMG and PANE mix more nodes from different classes. A similar observation can be made on Yelp, shown in the 2nd row of Figure 5.12. The visualization demonstrates the effectiveness of our methods in generating $\mathcal{L}$ for high-quality embedding.

5.6 Summary

This chapter introduces a spectrum-guided integration scheme for multi-view attributed graphs (MVAGs), enabling the direct application of classic graph algorithms for clustering and embedding tasks. The proposed approach emphasizes preserving community structure and node connectivity by leveraging the spectral properties of Laplacian matrices. To address computational challenges, we developed `SGLA`, an algorithm that delivers superior performance, followed by `SGLA+`, an accelerated variant that approximates the objective for improved efficiency. Extensive experiments demonstrate that both `SGLA` and `SGLA+` consistently produce high-quality results across diverse MVAG scenarios.

Chapter 6

Conclusion

6.1 Summary of Contributions

This thesis has advanced the field of attributed network analysis by developing innovative approaches to clustering and embedding that enhance the effectiveness, efficiency, and scalability of analyzing complex network structures. Through three interrelated contributions, we have addressed the challenges of attributed graphs, hypergraphs, and multi-view attributed graphs, delivering tailored solutions that integrate structural and attribute information. These advancements, validated through extensive experiments on real-world datasets, provide robust tools for applications in bioinformatics, social network analysis, and e-commerce, aligning with the thesis's goal of advancing clustering and embedding for attributed network structures.

The first contribution, the `ANCKA` framework [73, 72], offers a versatile and efficient solution for clustering attributed networks, encompassing attributed hypergraph clustering (AHC), attributed graph clustering (AGC), and attributed multiplex graph clustering (AMGC). By leveraging a KNN augmentation strategy, a novel random walk-based problem formulation, and an optimized iterative framework, `ANCKA` achieves superior cluster quality while maintaining high efficiency. To further enhance scalability, we developed ANCKA-GPU, a GPU-accelerated variant that outperforms its CPU-parallel counterpart on large datasets while preserving effectiveness. Extensive evaluations across real-world datasets demonstrate ANCKA 's outstanding performance, setting a new benchmark for clustering diverse attributed networks.

The second contribution introduced SAHE, an efficient algorithm for attributed hypergraph node and hyperedge embedding (AHNEE). SAHE generates node and hyperedge embeddings that preserve higher-order connectivities and attribute similarities in attributed hypergraphs, utilizing multi-hop similarity measures (HMS-N and HMS-E) and optimized decomposition techniques. With log-linear time complexity, SAHE outperforms 11 baselines across 8 real-world datasets, demonstrating both scalability and effectiveness. Its design makes it well-suited for applications such as genomic expression modeling and online shopping behavior prediction, advancing representation learning for higher-order network structures.

The third contribution developed a spectrum-guided integration scheme for clustering and embedding in multi-view attributed graphs (MVAGs), implemented through two efficient algorithms, SGLA and SGLA+. By formulating a constrained optimization problem over a joint objective function that preserves community structure and node connectivity via the Laplacian matrix spectrum, this scheme enables existing graph algorithms to be applied to MVAGs. SGLA delivers superior performance compared to baselines, while SGLA+ further reduces computational demands through approximation, maintaining high-quality results. Extensive experiments confirm their effectiveness and efficiency, supporting applications like recommendation systems and neuroimaging analysis.

Collectively, these contributions, namely ANCKA, SAHE, and SGLA/SGLA+, form a cohesive set of frameworks that advance clustering and embedding by addressing the unique complexities of attributed network structures. Their validated performance across diverse datasets underscores their potential to transform network analysis in real-world settings.

6.2 Implications and Impact

The methods presented in this thesis have significant implications for both theoretical and applied network analysis. Theoretically, our work advances the approach to attributed network clustering by demonstrating that a unified framework can effectively handle the diversity of attributed networks. ANCKA 's random walk-based formulation and KNN augmentation depart from traditional clustering paradigms, offering a generalizable model for graphs, hypergraphs, and multiplex graphs [144]. SAHE 's log-linear time embedding approach expands the range of factorization-based embedding methods to hypergraphs, addressing a gap in hypergraph modeling [157]. The spectrum-guided integration of SGLA and SGLA+ provides a novel framework for multi-view attributed graph analysis, unifying clustering and embedding tasks through the spectral properties of the Laplacian matrix [153]. These advancements enrich the theoretical foundations of network analysis, paving the way for future methodological innovations.

Practically, our contributions have solid potential for applications to real-world network data. The scalability of the proposed approaches, exemplified by ANCKA-GPU 's GPU acceleration and SAHE 's log-linear complexity, ensures their utility for large-scale networks, addressing computational bottlenecks noted in prior works. The interdisciplinary outreach of our frameworks further amplifies their impact across multiple domains, including bioinformatics, social network analysis, and recommendation systems. For instance, clustering academic hypergraphs by co-authorship and keywords [27] supports research collaboration analysis, while multi-view clustering aids neuroimaging for disease diagnosis [153]. These diverse applications highlight the significance of the research topic and the versatility of our approaches, meeting the growing demand for advanced, scalable network analysis tools in data-driven areas.

6.3 Future Directions

This thesis opens several promising avenues for future research. For ANCKA, we may extend the framework to handle evolving attributed networks and implement distributed computations to enhance scalability. Developing robust augmentation strategies for highly noisy or sparse datasets would further improve its versatility. For SAHE, enhancing the algorithm with incremental decomposition methods for dynamic hypergraphs could support applications where incorporating the latest updates is essential. This embedding approach may also be extended to other attributed networks, and a GPU implementation could further boost its computational efficiency. For SGLA and SGLA+, future work could focus on handling dynamic MVAGs through lazy update schemes and incremental objective evaluation to minimize update costs. Designing techniques for robustness against noisy MVAGs and exploring GPU computation for multi-view analysis would enhance their practical utility.

6.4 Concluding Remarks

This thesis has made substantial contributions to the analysis of attributed network structures, delivering frameworks—ANCKA, SAHE, and SGLA/SGLA+ —that advance clustering and embedding for graphs, hypergraphs, and multi-view graphs. Through innovative techniques like KNN augmentation, efficient approximation, and spectrum-guided integration, our work achieves high-quality outcomes with computational efficiency, as validated across diverse real-world datasets. The theoretical and practical implications span a wide range of areas, demonstrating the power of tailored and unified approaches. Looking forward, they inspire exciting future directions, from dynamic network analysis to interdisciplinary applications. Ultimately, this thesis advances the field of network analysis, providing tools and insights that empower researchers and practitioners to interpret and exploit attributed networks.

References

[1] Zeyuan Allen Zhu, Silvio Lattanzi, and Vahab Mirrokni. A local algorithm for finding well-connected clusters. In *ICML*, 2013.

[2] Konstantinos Ameranis, Adela Frances DePavia, Lorenzo Orecchia, and Erasmo Tani. Fast Algorithms for Hypergraph PageRank with Applications to Semi-Supervised Learning. In *ICML*, June 2024.

[3] Jianjing An and Dong Wang. Efficient one-sided jacobi svd computation on amd gpu using opencl. In *ICSP*, 2016.

[4] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *FOCS*, pages 475–486, 2006.

[5] Haitham Ashoor, Xiaowen Chen, Wojciech Rosikiewicz, Jiahui Wang, Albert Cheng, Ping Wang, Yijun Ruan, and Sheng Li. Graph embedding and unsupervised learning predict genomic sub-compartments from HiC chromatin interaction data. *Nature Communications*, 11(1):1173, March 2020.

[6] Ali Behrouz, Farnoosh Hashemi, and Laks V. S. Lakshmanan. Firmtruss community search in multilayer networks. *Proc. VLDB Endow.*, 16(3):505–518, 2022.

[7] Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon Kleinberg. Simplicial closure and higher-order link prediction. *PNAS*, 115(48):E11221–E11230, 2018.

[8] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008.

[9] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *KDD*, pages 2464–2473. ACM, 2020.

[10] Deng Cai, Xiaofei He, Jiawei Han, and Thomas S Huang. Graph regularized nonnegative matrix factorization for data representation. *TPAMI*, 33(8):1548–1560, 2010.

[11] Yaoming Cai, Zijia Zhang, Zhihua Cai, Xiaobo Liu, and Xinwei Jiang. Hypergraph-structured autoencoder for unsupervised and semisupervised classification of hyperspectral image. *IEEE GRSL*, 19:1–5, 2022.

[12] T.-H. Hubert Chan and Zhibin Liang. Generalizing the Hypergraph Laplacian via a Diffusion Process with Mediators. *ArXiv*, abs/1804.11128, 2018.

[13] Hong Cheng, Yang Zhou, and Jeffrey Xu Yu. Clustering large attributed graphs: A balance between structural and attribute similarities. *ACM TKDD*, 5(2):1–33, 2011.

[14] Jiafeng Cheng, Qianqian Wang, Zhiqiang Tao, De-Yan Xie, and Quanxue Gao. Multi-View Attribute Graph Convolution Networks for Clustering. In *IJCAI*, pages 2973–2979, 2020.

[15] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*, 2019.

[16] Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. You are AllSet: A Multiset Function Framework for Hypergraph Neural Networks. In *ICLR*, 2021.

[17] Uthsav Chitra and Benjamin Raphael. Random Walks on Hypergraphs with Edge-Dependent Vertex Weights. In *ICML*, pages 1172–1181, 2019.

[18] Fan RK Chung. *Spectral Graph Theory*, volume 92. American Mathematical Soc., 1997.

[19] Shane Cook. *CUDA programming: a developer's guide to parallel computing with GPUs*. Newnes, 2012.

[20] Daryl R. DeFord and Scott D. Pauls. Spectral clustering methods for multiplex networks. *ArXiv*, abs/1703.05355, 2017.

[21] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library, 2024.

[22] Boxin Du, Changhe Yuan, Robert Barton, Tal Neiman, and Hanghang Tong. Self-supervised Hypergraph Representation Learning. In *IEEE Big Data*, pages 505–514, December 2022.

[23] Rundong Du, Barry Drake, and Haesun Park. Hybrid clustering based on content and connection structure using joint nonnegative matrix factorization. *J. Glob. Optim.*, 74(4):861–877, 2019.

[24] Shaohua Fan, Xiao Wang, Chuan Shi, Emiao Lu, Ken Lin, and Bai Wang. One2multi graph autoencoder for multi-view graph clustering. In *WWW*, pages 3070–3076, 2020.

[25] Uno Fang, Man Li, Jianxin Li, Longxiang Gao, Tao Jia, and Yanchun Zhang. A comprehensive survey on multi-view clustering. *IEEE TKDE.*, 35(12):12350–12368, 2023.

[26] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. Effective community search for large attributed graphs. *Proc. VLDB Endow.*, 9(12):1233–1244, 2016.

[27] Barakeel Fanseu Kamhoua, Lin Zhang, Kaili Ma, James Cheng, Bo Li, and Bo Han. HyperGraph Convolution Based Attributed HyperGraph Clustering. In *CIKM*, pages 453–463, 2021.

[28] Song Feng, Emily Heath, Brett Jefferson, Cliff Joslyn, Henry Kvinge, Hugh D. Mitchell, Brenda Praggastis, Amie J. Eisfeld, Amy C. Sims, Larissa B. Thackray, Shufang Fan, Kevin B. Walters, Peter J. Halfmann, Danielle Westhoff-Smith, Qing Tan, Vineet D. Menachery, Timothy P. Sheahan, Adam S. Cockrell, Jacob F. Kocher, Kelly G. Stratton, Natalie C. Heller, Lisa M. Bramer, Michael S. Diamond, Ralph S. Baric, Katrina M. Waters, Yoshihiro Kawaoka, Jason E. McDermott, and Emilie Purvine. Hypergraph models of biological networks to identify genes critical to pathogenic viral response. *BMC Bioinformatics*, 22(1):287, May 2021.

[29] Zijin Feng, Miao Qiao, Chengzhi Piao, and Hong Cheng. On Graph Representation for Attributed Hypergraph Clustering. *Proc. ACM Manag. Data*, 3(1):59:1–59:26, February 2025.

[30] Chakib Fettal, Lazhar Labiod, and Mohamed Nadif. Simultaneous linear multi-view attributed graph representation learning and clustering. In *WSDM*, pages 303–311. ACM, 2023.

[31] Santo Fortunato. Community detection in graphs. *ArXiv*, abs/0906.0612, 2009.

[32] Haitao Fu, Feng Huang, Xuan Liu, Yang Qiu, and Wen Zhang. MVGCN: data integration through multi-view graph convolutional network for predicting links in biomedical bipartite networks. *Bioinform.*, 38(2):426–434, 2022.

[33] Yue Gao, Yifan Feng, Shuyi Ji, and Rongrong Ji. HGNN+: General Hypergraph Neural Networks. *TPAMI*, 45(3):3181–3199, March 2023.

[34] Thomas Gaudelet, Noël Malod-Dognin, and Natasa Przulj. Higher-order molecular organization as a source of biological function. *Bioinformatics*, 34(17):i944–i953, 2018.

[35] Lars Gottesbüren, Tobias Heuer, and Peter Sanders. Parallel flow-based hypergraph partitioning. In *SEA*, volume 233, 2022.

[36] Aditya Grover and Jure Leskovec. Node2vec: Scalable Feature Learning for Networks. In *KDD*, pages 855–864, August 2016.

[37] Roger Guimerà and Luis A. Nunes Amaral. Functional cartography of complex metabolic networks. *Nature*, 433:895–900, 2005.

[38] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *ICML*, 2020.

[39] Insu Han, Haim Avron, and Jinwoo Shin. Polynomial Tensor Sketch for Element-wise Function of Low-Rank Matrix. In *ICML*, pages 3984–3993, 2020.

[40] Yan Han, Edward W. Huang, Wenqing Zheng, Nikhil Rao, Zhangyang Wang, and Karthik Subbian. Search Behavior Prediction: A Hypergraph Perspective. In *WSDM*, WSDM '23, pages 697–705, February 2023.

[41] Koby Hayashi, Sinan G. Aksoy, Cheong Hee Park, and Haesun Park. Hypergraph random walks, laplacians, and clustering. In *CIKM*, pages 495–504, 2020.

[42] Michael T. Heath. *Scientific Computing: An Introductory Survey, Revised Second Edition.* Society for Industrial and Applied Mathematics, 2018.

[43] Matthias Hein, Simon Setzer, Leonardo Jost, and Syama Sundar Rangapuram. The Total Variation on Hypergraphs - Learning on Hypergraphs Revisited. In *NeurIPS*, volume 26, 2013.

[44] Jie Huang, Chuan Chen, Fanghua Ye, Weibo Hu, and Zibin Zheng. Nonuniform Hyper-Network Embedding with Dual Mechanism. *ACM Trans. Inf. Syst.*, 38(3):28:1–28:18, May 2020.

[45] Jie Huang, Chuan Chen, Fanghua Ye, Jiajing Wu, Zibin Zheng, and Guohui Ling. Hyper2vec: Biased Random Walk for Hyper-network Embedding. In *DASFAA*, pages 273–277, 2019.

[46] Ling Huang, Chang-Dong Wang, and Philip S. Yu. Higher Order Connection Enhanced Community Detection in Adversarial Multiview Networks. *IEEE Trans. Cybern.*, pages 1–15, 2021.

[47] Wentao Huang, Yuchen Li, Yuan Fang, Ju Fan, and Hongxia Yang. BiANE: Bipartite Attributed Network Embedding. In *SIGIR*, SIGIR '20, pages 149–158, July 2020.

[48] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *WWW*, pages 271–279, 2003.

[49] Baoyu Jing, Chanyoung Park, and Hanghang Tong. HDMI: high-order deep multiplex infomax. In *WWW*, pages 2414–2424, 2021.

[50] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE TBD*, 7(3):535–547, 2019.

[51] Jinhong Jung, Namyong Park, Sael Lee, and U Kang. Bepi: Fast and memory-efficient method for billion-scale random walk with restart. In *SIGMOD*, page 789–804, 2017.

[52] Barakeel Fanseu Kamhoua, Lin Zhang, Kaili Ma, James Cheng, Boyang Li, and Bo Han. Grace: A general graph convolution framework for attributed graph clustering. *ACM TKDD*, 17:1 – 31, 2022.

[53] Zhao Kang, Zhanyu Liu, Shirui Pan, and Ling Tian. Fine-grained attributed graph clustering. In *SDM*, 2022.

[54] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Applications in VLSI domain. *IEEE TVLSI*, 7(1):69–79, 1999.

[55] George Karypis and Vipin Kumar. Multilevel k-way Partitioning Scheme for Irregular Graphs. *J. Parallel Distributed Comput.*, 48(1):96–129, 1998.

[56] Tosio Kato. *Perturbation Theory for Linear Operators*, volume 132. Springer Science & Business Media, 2013.

[57] Aparajita Khan and Pradipta Maji. Approximate graph laplacians for multimodal data clustering. *IEEE TPAMI.*, 43(3):798–813, 2021.

[58] Muhammad Raza Khan and Joshua E. Blumenstock. Multi-gcn: Graph convolutional networks for multi-view networks, with applications to global poverty. In *AAAI*, pages 606–613, 2019.

[59] Shima Khoshraftar and Aijun An. A Survey on Graph Representation Learning Methods. *ACM Trans. Intell. Syst. Technol.*, 15(1):19:1–19:55, January 2024.

[60] Sunwoo Kim, Shinhwan Kang, Fanchen Bu, Soo Yong Lee, Jaemin Yoo, and Kijung Shin. HypeBoy: Generative Self-Supervised Representation Learning on Hypergraphs. In *ICLR*, October 2023.

[61] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[62] Tarun Kumar, Sankaran Vaidyanathan, Harini Ananthapadmanabhan, Srinivasan Parthasarathy, and Balaraman Ravindran. Hypergraph Clustering: A Modularity Maximization Approach. *ArXiv*, abs/1812.10869, 2018.

[63] Kevin Lang and Satish Rao. A flow-based method for improving the expansion or conductance of graph cuts. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 325–337, 2004.

[64] Dongjin Lee and Kijung Shin. I'm Me, We're Us, and I'm Us: Tri-directional Contrastive Learning on Hypergraphs. *AAAI*, 37(7):8456–8464, June 2023.

[65] Geon Lee, Soo Yong Lee, and Kijung Shin. VilLain: Self-Supervised Learning on Homogeneous Hypergraphs without Features via Virtual Label Propagation. In *WWW*, pages 594–605, May 2024.

[66] James R. Lee, Shayan Oveis Gharan, and Luca Trevisan. Multiway spectral partitioning and higher-order cheeger inequalities. *J. ACM*, 61(6):37:1–37:30, 2014.

[67] R. B. Lehoucq and D. C. Sorensen. Deflation Techniques for an Implicitly Restarted Arnoldi Iteration. *SIAM Journal on Matrix Analysis and Applications*, 17(4):789–821, 1996.

[68] Chaozhuo Li, Senzhang Wang, Lifang He, Philip S. Yu, Yanbo Liang, and Zhoujun Li. SSDMV: semi-supervised deep social spammer detection by multi-view data fusion. In *ICDM*, pages 247–256, 2018.

[69] Shuyang Li, Yufei Li, Jianmo Ni, and Julian McAuley. SHARE: A System for Hierarchical Assistive Recipe Editing. In *EMNLP*, pages 11077–11090, 2022.

[70] Yiran Li, Gongyao Guo, Jieming Shi, Sibo Wang, and Qing Li. Efficient Integration of Multi-View Attributed Graphs for Clustering and Embedding . In *ICDE*, pages 3863–3875, 2025.

[71] Yiran Li, Gongyao Guo, Jieming Shi, Renchi Yang, Shiqi Shen, Qing Li, and Jun Luo. Technical report, 2024. `https://sites.google.com/view/ancka-technical-report/`.

[72] Yiran Li, Gongyao Guo, Jieming Shi, Renchi Yang, Shiqi Shen, Qing Li, and Jun Luo. A versatile framework for attributed network clustering via K-nearest neighbor augmentation. *VLDB J.*, 33(6):1913–1943, 2024.

[73] Yiran Li, Renchi Yang, and Jieming Shi. Efficient and Effective Attributed Hypergraph Clustering via K-Nearest Neighbor Augmentation. *Proc. ACM Manag. Data*, 1(2):116:1–116:23, June 2023.

[74] Zhonghang Li, Chao Huang, Lianghao Xia, Yong Xu, and Jian Pei. Spatial-Temporal Hypergraph Self-Supervised Learning for Crime Prediction. In *ICDE*, pages 2984–2996, 2022.

[75] Bei Lin, You Li, Ning Gui, Zhuopeng Xu, and Zhiwu Yu. Multi-view graph representation learning beyond homophily. *ACM TKDD*, 17(8):114:1–114:21, 2023.

[76] Zhiping Lin and Zhao Kang. Graph filter-based multi-view attributed graph clustering. In *IJCAI*, pages 2723–2729, 2021.

[77] Zhiping Lin, Zhao Kang, Lizong Zhang, and Ling Tian. Multi-view attributed graph clustering. *IEEE TKDE.*, 35(2):1872–1880, 2023.

[78] Yue Liu, Jun Xia, Sihang Zhou, Xihong Yang, Ke Liang, Chenchen Fan, Yan Zhuang, Stan Z. Li, Xinwang Liu, and Kunlun He. A Survey of Deep Graph Clustering: Taxonomy, Challenge, Application, and Open Resource, September 2023.

[79] Yunfei Liu, Zhen Liu, Xiaodong Feng, and Zhongyi Li. Robust attributed network embedding preserving community information. In *ICDE*, pages 1874–1886. IEEE, 2022.

[80] S. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–137, 1982.

[81] Yuanfu Lu, Chuan Shi, Linmei Hu, and Zhiyuan Liu. Relation structure-aware heterogeneous information network embedding. In *AAAI*, pages 4456–4463, 2019.

[82] Khanh Luong and Richi Nayak. A novel approach to learning consensus and complementary information for multi-view data clustering. In *IEEE ICDE*, pages 865–876, 2020.

[83] Peter Macgregor. Fast and simple spectral clustering in theory and practice. *Advances in Neural Information Processing Systems*, 36, 2024.

[84] Federico Magliani and Andrea Prati. LSH kNN graph for diffusion on image retrieval. *Information Retrieval Journal*, 24(2):114–136, 2021.

[85] Meta. Meta Reports Fourth Quarter and Full Year 2024 Results. https://investor.atmeta.com/investor-news/press-release-details/2025/Meta-Reports-Fourth-Quarter-and-Full-Year-2024-Results/, 2025. [Online; accessed 12-April-2025].

[86] Yujie Mo, Yajie Lei, Jialie Shen, Xiaoshuang Shi, Heng Tao Shen, and Xiaofeng Zhu. Disentangled multiplex graph representation learning. In *ICML*, volume 202, pages 24983–25005, 2023.

[87] Nairouz Mrabah, Mohamed Bouguessa, Mohamed Fawzi Touati, and Riadh Ksantini. Rethinking graph auto-encoder models for attributed graph clustering (extended abstract). In *IEEE ICDE*, pages 3891–3892, 2023.

[88] Hiroshi Nagamochi and Toshihide Ibaraki. *Algorithmic Aspects of Graph Connectivity*, volume 123 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2008.

[89] M. E. J. Newman. Spectral methods for community detection and graph partitioning. *Physical Review E*, 88(4):042822, October 2013.

[90] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP-IJCNLP*, pages 188–197, 2019.

[91] Erlin Pan and Zhao Kang. Multi-view contrastive graph clustering. In *NeurIPS*, pages 2148–2159, 2021.

[92] Chanyoung Park, Donghyun Kim, Jiawei Han, and Hwanjo Yu. Unsupervised Attributed Multiplex Network Embedding. *AAAI*, 2020.

[93] Haekyu Park, Jinhong Jung, and U. Kang. A comparative study of matrix factorization and random walk with restart in recommender systems. In *IEEE BigData*, pages 756–765, 2017.

[94] Prasanna Patil, Govind Sharma, and M Narasimha Murty. Negative sampling for hyperlink prediction in networks. In *PAKDD*, pages 607–619. Springer, 2020.

[95] Liang Peng, Xin Wang, and Xiaofeng Zhu. Unsupervised multiplex graph learning with complementary and consistent information. In *ACM MM*, pages 454–462, 2023.

[96] Marianna Pensky and Yaxuan Wang. Clustering of diverse multiplex networks. *ArXiv*, abs/2110.05308, 2021.

[97] MJD Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. *Advances in Optimization and Numerical Analysis*, pages 51–67, 1994.

[98] Jiezhong Qiu, Laxman Dhulipala, Jie Tang, Richard Peng, and Chi Wang. LightNE: A Lightweight Graph Processing System for Network Embedding. In *SIGMOD*, pages 2281–2289, June 2021.

[99] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *WSDM*, pages 459–467, 2018.

[100] Tom M. Ragonneau and Zaikun Zhang. An optimal interpolation set for model-based derivative-free optimization methods. *Optimization Methods and Software*, pages 1–13, 2024.

[101] Matthew J. Rattigan, Marc Maier, and David Jensen. Graph clustering with network structure indices. In *ICML*, page 783–790, 2007.

[102] J.A. Rodri ́guez. On the Laplacian Eigenvalues and Metric Parameters of Hypergraphs. *Linear Multilinear Algebra*, 50(1):1–14, 2002.

[103] M. Rosvall, D. Axelsson, and C. T. Bergstrom. The map equation. *The European Physical Journal Special Topics*, 178(1):13–23, 2009.

[104] Yousef Saad. *Numerical methods for large eigenvalue problems: revised edition.* SIAM, 2011.

[105] Sebastian Schlag, Tobias Heuer, Lars Gottesbüren, Yaroslav Akhremtsev, Christian Schulz, and Peter Sanders. High-quality hypergraph partitioning. *J. Exp. Algorithmics*, 2022.

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

[107] Jieming Shi, Nikos Mamoulis, Dingming Wu, and David W. Cheung. Density-based place clustering in geo-social networks. In *SIGMOD Conference*, pages 99–110. ACM, 2014.

[108] Jieming Shi, Renchi Yang, Tianyuan Jin, Xiaokui Xiao, and Yin Yang. Realtime top-k personalized pagerank over large graphs on gpus. *VLDB*, 13(1):15–28, 2019.

[109] Shashank Sheshar Singh, Samya Muhuri, Shivansh Mishra, Divya Srivastava, Harish Kumar Shakya, and Neeraj Kumar. Social Network Analysis: A Survey on Process, Tools, and Application. *ACM Comput. Surv.*, 56(8):192:1–192:39, April 2024.

[110] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Paul Hsu, and Kuansan Wang. An overview of microsoft academic service (MAS) and applications. In *WWW*, pages 243–246, 2015.

[111] Daniel Spielman. Spectral graph theory. *Combinatorial scientific computing*, 18:18, 2012.

[112] Daniel A. Spielman. Spectral graph theory and its applications. In *IEEE FOCS.*, pages 29–38, 2007.

[113] Yuuki Takai, Atsushi Miyauchi, Masahiro Ikeda, and Yuichi Yoshida. Hypergraph clustering based on pagerank. In *KDD*, 2020.

[114] Qiaoyu Tan, Xin Zhang, Xiao Huang, Hao Chen, Jundong Li, and Xia Hu. Collaborative Graph Neural Networks for Attributed Network Embedding. *IEEE TKDE.*, 36(3):972–986, 2024.

[115] Riitta Toivonen, Lauri Kovanen, Mikko Kivelä, Jukka-Pekka Onnela, Jari Saramäki, and Kimmo Kaski. A comparative study of social network models: Network evolution models and nodal attribute models. *Social Networks*, 31(4):240–254, 2009.

[116] Hanghang Tong, Christos Faloutsos, and Jia-yu Pan. Fast random walk with restart and its applications. In *ICDM*, pages 613–622, 2006.

[117] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan Oseledets, and Emmanuel Müller. FREDE: Anytime graph embeddings. *VLDB*, 14(6):1102–1110, February 2021.

[118] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *JMLR*, 9(11), 2008.

[119] Menghan Wang, Yujie Lin, Guli Lin, Keping Yang, and Xiao-Ming Wu. M2GRL: A multi-task multi-view graph representation learning framework for web-scale recommender systems. In *KDD*, pages 2349–2358. ACM, 2020.

[120] Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and Philip S. Yu. A Survey on Heterogeneous Graph Embedding: Methods, Techniques, Applications and Sources. *IEEE Transactions on Big Data*, 9(2):415–436, April 2023.

[121] Joe H. Ward Jr. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[122] Tianxin Wei, Yuning You, Tianlong Chen, Yang Shen, Jingrui He, and Zhangyang Wang. Augmentations in Hypergraph Contrastive Learning: Fabricated and Generative. *NeurIPS*, 35:1909–1922, December 2022.

[123] Joyce Jiyoung Whang, Rundong Du, Sangwon Jung, Geon Lee, Barry L. Drake, Qingqing Liu, Seonggoo Kang, and Haesun Park. MEGA: multi-view semi-supervised clustering of hypergraphs. *Proc. VLDB Endow.*, 13(5):698–711, 2020.

[124] Joong-Ho Won, Hua Zhou, and Kenneth Lange. Orthogonal trace-sum maximization: Applications, local algorithms, and global optimality. *SIAM J. Matrix Anal. Appl.*, 42(2):859–882, 2021.

[125] Anbiao Wu, Ye Yuan, Changsheng Li, Yuliang Ma, and Hao Zhang. Attributed Network Embedding in Streaming Style. In *ICDE*, pages 3138–3150, May 2024.

[126] Lei Wu, Yufeng Hu, Yajin Zhou, Haoyu Wang, Xiapu Luo, Zhi Wang, Fan Zhang, and Kui Ren. Towards understanding and demystifying bitcoin mixing services. In *WWW*, pages 33–44, 2021.

[127] Ming-Juan Wu, Ying-Lian Gao, Jin-Xing Liu, Chun-Hou Zheng, and Juan Wang. Integrative hypergraph regularization principal component analysis for sample clustering and co-expression genes network analysis on multi-omics data. *IEEE JBHI*, 24(6), 2020.

[128] Xueyi Wu, Yuanyuan Xu, Wenjie Zhang, and Ying Zhang. Billion-Scale Bipartite Graph Embedding: A Global-Local Induced Approach. *VLDB*, 17(2):175–183, October 2023.

[129] Zhourun Wu, Mingyue Guo, Xiaopeng Jin, Junjie Chen, and Bin Liu. CFAGO: Cross-fusion of network and attributes based on attention mechanism for protein function prediction. *Bioinformatics*, 39(3):btad123, March 2023.

[130] Lianghao Xia, Chao Huang, and Chuxu Zhang. Self-Supervised Hypergraph Transformer for Recommender Systems. In *KDD*, pages 2100–2109, August 2022.

[131] Yuyang Xie, Yuxiao Dong, Jiezhong Qiu, Wenjian Yu, Xu Feng, and Jie Tang. SketchNE: Embedding Billion-Scale Networks Accurately in One Hour. *TKDE*, pages 1–14, 2023.

[132] Mengjia Xu. Understanding Graph Embedding Methods and Their Applications. *SIAM Review*, 63(4):825–853, January 2021.

[133] Rongwei Xu, Guanfeng Liu, Yan Wang, Xuyun Zhang, Kai Zheng, and Xiaofang Zhou. Adaptive Hypergraph Network for Trust Prediction. In *ICDE*, pages 2986–2999, 2024.

[134] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. A model-based approach to attributed graph clustering. In *SIGMOD*, pages 505–516, 2012.

[135] Amitai Yacobi, Ofir Lindenbaum, and Uri Shaham. Generalizable and robust spectral method for multi-view representation learning, 2025.

[136] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. HyperGCN: a new method of training graph convolutional networks on hypergraphs. In *NeurIPS*, pages 1511–1522, 2019.

[137] Yuguang Yan, Yuanlin Chen, Shibo Wang, Hanrui Wu, and Ruichu Cai. Hypergraph Joint Representation Learning for Hypervertices and Hyperedges via Cross Expansion. *AAAI*, 38(8):9232–9240, March 2024.

[138] Dingqi Yang, Bingqing Qu, Jie Yang, and Philippe Cudré-Mauroux. LBSN2Vec++: Heterogeneous Hypergraph Embedding for Location-Based Social Networks. *TKDE*, 34(4):1843–1855, April 2022.

[139] Jaewon Yang, Julian McAuley, and Jure Leskovec. Community detection in networks with node attributes. In *ICDM*, pages 1151–1156, 2013.

[140] Mingji Yang, Hanzhi Wang, Zhewei Wei, Sibo Wang, and Ji-Rong Wen. Efficient Algorithms for Personalized PageRank Computation: A Survey. *TKDE*, 36(9):4582–4602, September 2024.

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

[142] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Sourav S. Bhowmick, and Juncheng Liu. PANE: Scalable and effective attributed network embedding. *VLDBJ*, 32(6):1237–1262, November 2023.

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

[144] Renchi Yang, Jieming Shi, Yin Yang, Keke Huang, Shiqi Zhang, and Xiaokui Xiao. Effective and Scalable Clustering on Massive Attributed Graphs. In *WWW*, pages 3675–3687, 2021.

[145] Tianbao Yang, Rong Jin, Yun Chi, and Shenghuo Zhu. Combining link and content for community detection: A discriminative approach. In *KDD*, pages 927–936, 2009.

[146] Yuan Yin and Zhewei Wei. Scalable Graph Embeddings via Sparse Transpose Proximities. In *KDD*, KDD '19, pages 1429–1437, July 2019.

[147] Chia-An Yu, Ching-Lun Tai, Tak-Shing Chan, and Yi-Hsuan Yang. Modeling Multi-way Relations with Hypergraph Embedding. In *CIKM*, CIKM '18, pages 1707–1710, October 2018.

[148] Stella X. Yu and Jianbo Shi. Multiclass Spectral Clustering. In *ICCV*, page 313, 2003.

[149] Xiang Yue, Zhen Wang, Jingong Huang, Srinivasan Parthasarathy, Soheil Moosavinasab, Yungui Huang, Simon M Lin, Wen Zhang, Ping Zhang, and Huan Sun. Graph embedding on biomedical networks: Methods, applications and evaluations. *Bioinformatics*, 36(4):1241–1251, February 2020.

[150] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM TALG*, 1(1):2–13, 2005.

[151] J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid. Local Features and Kernels for Classification of Texture and Object Categories: A Comprehensive Study. *International Journal of Computer Vision*, 73(2):213–238, 2007.

[152] Rui Zhang, Arthur Zimek, and Peter Schneider-Kamp. Unsupervised representation learning on attributed multiplex network. In *CIKM*, pages 2610–2619, 2022.

[153] Xi Zhang, Lifang He, Kun Chen, Yuan Luo, Jiayu Zhou, and Fei Wang. Multi-View Graph Convolutional Network and Its Applications on Neuroimage Analysis for Parkinson's Disease. *AMIA Annual Symposium Proceedings*, 2018:1147–1156, December 2018.

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

[155] Xiangyu Zhao, Maolin Wang, Xinjian Zhao, Jiansheng Li, Shucheng Zhou, Dawei Yin, Qing Li, Jiliang Tang, and Ruocheng Guo. Embedding in recommender systems: A survey, 2023.

[156] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with Hypergraphs: Clustering, Classification, and Embedding. In *NeurIPS*, volume 19, 2007.

[157] Jingya Zhou, Ling Liu, Wenqi Wei, and Jianxi Fan. Network Representation Learning: From Preprocessing, Feature Extraction to Node Embedding. *ACM Comput. Surv.*, 55(2):38:1–38:35, January 2022.

[158] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. Graph clustering based on structural/attribute similarities. *Proceedings of the VLDB Endowment*, 2(1):718–729, 2009.

[159] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. Clustering Large Attributed Graphs: An Efficient Incremental Approach. In *ICDM*, pages 689–698, 2010.

[160] Linlin Zong, Xianchao Zhang, Xinyue Liu, and Hong Yu. Weighted multi-view spectral clustering based on spectral perturbation. In *AAAI*, pages 4621–4629, 2018.