Effective and Efficient Attributed Hypergraph Embedding on Nodes and Hyperedges [Technical Report]

Yiran Li, Gongyao Guo, Chen Feng, Jieming Shi*
The Hong Kong Polytechnic University
Hong Kong SAR, China
{yi-ran.li,gongyao.guo}@connect.polyu.hk;{chenfeng,jieming.shi}@polyu.edu.hk

ABSTRACT

An attributed hypergraph comprises nodes with attributes and hyperedges that connect varying numbers of nodes. *Attributed hypergraph node and hyperedge embedding* (AHNEE) maps nodes and hyperedges to compact vectors for use in important tasks such as node classification, hyperedge link prediction, and hyperedge classification. Generating high-quality embeddings is challenging due to the complexity of attributed hypergraphs and the need to embed both nodes and hyperedges, especially in large-scale data. Existing solutions often fall short by focusing only on nodes or lacking native support for attributed hypergraphs, leading to inferior quality, and struggle with scalability on large attributed hypergraphs.

We propose SAHE, an efficient and effective approach that unifies node and hyperedge embeddings for AHNEE computation, advancing the state of the art via comprehensive embedding formulations and algorithmic designs. First, we introduce two higher-order similarity measures, HMS-N and HMS-E, to capture similarities between node pairs and hyperedge pairs, respectively. These measures consider multi-hop connections and global topology within an extended hypergraph that incorporates attribute-based hyperedges. SAHE formulates the AHNEE objective to jointly preserve all-pair HMS-N and HMS-E similarities. Direct optimization is computationally expensive, so we analyze and unify core approximations of all-pair HMS-N and HMS-E to solve them simultaneously. To enhance efficiency, we design several non-trivial optimizations that avoid iteratively materializing large dense matrices while maintaining high-quality results. Extensive experiments on diverse attributed hypergraphs and 3 downstream tasks, compared against 11 baselines, show that SAHE consistently outperforms existing methods in embedding quality and is up to orders of magnitude faster.

PVLDB Reference Format:
Yiran Li, Gongyao Guo, Chen Feng, Jieming Shi. Effective and Efficient Attributed Hypergraph Embedding on Nodes and Hyperedges [Technical Report]. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/CyanideCentral/AHNEE.

*Corresponding Author.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

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 higher-order relationships among entities, e.g., social networks [2], genomic expression [7], and online shopping sessions [12]. 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 [46] and industry [34], and has been studied on various types of simple graphs with pairwise edge connections, such as homogeneous graphs [25, 29, 40] and attributed graphs [31, 42]. 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 capture the connections and attribute semantics of groups of nodes around hyperedges. The embeddings are valuable for downstream tasks: node embeddings facilitate node classification [14] and hyperedge link prediction [44], while hyperedge embeddings support hyperedge classification [37].

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-range 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. As reviewed in Section 6, an early study [45] uses the hypergraph Laplacian spectrum for node embeddings, while [13, 14] extend graph-based node embedding to hypergraphs. However, these approaches typically do not consider attribute information or hyperedge embedding generation, with some, like [45], overlooking long-range connectivity. A

recent class of studies [16, 17, 30] 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 [32, 42]. 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 Scalable Attributed Hypergraph node and hyperedge Embedding 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 10 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 pairs 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 diverse real datasets and 3 downstream tasks demonstrate the effectiveness and efficiency of our method.

Table 1: Frequently used notations.

Notation	Description
$H = \{\mathcal{V}, \mathcal{E}, \mathbf{X}\}$	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 $\lvert\mathcal{E}\rvert = m$.
$d(v), \delta(e)$	The generalized degree of a node v and 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.

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 1 gives an 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 H, $\mathrm{vol}(H) = \sum_{v \in \mathcal{V}} d(v)$, measures its total weighted connectivity by summing the generalized node 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: *node classification* and *hyperedge link prediction* with node embeddings; *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 1 lists the frequently used notations in our paper.

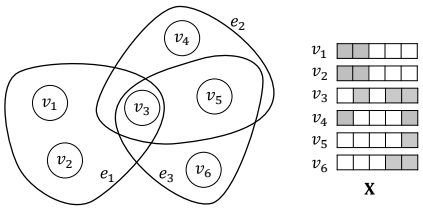
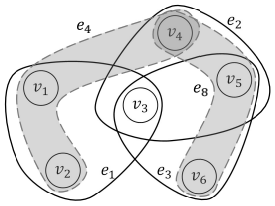


Figure 1: An example of attributed hypergraph H.



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

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

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

Figure 2: Extended hypergraph $\mathcal{H}$.

3 SIMILARITIES AND OBJECTIVES

As explained in Section 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 3.1. Then in Section 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 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 3.4 presents a preliminary method to directly solve this problem.

3.1 Attribute-Extended Hypergraph

The literature on attributed data [8, 21] 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 by attribute similarity. We adopt this approach to construct an extended hypergraph $\mathcal{H}$ from the input H, with dedicated designs tailored for hypergraph structures while 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 1, we construct five attribute-based hyperedges $\{e_4, \ldots, e_8\}$ for nodes $v_1, \ldots, v_5$, respectively. With $K = 2$, Figure 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 $\text{vol}(\mathcal{E}) = \sum_{e \in \mathcal{E}} w(e)\delta(e)$, which can influence similarity measures by skewing transitions to $\mathcal{E}$ or $\mathcal{E}_K$, affecting embedding priorities. Thus, we balance structural and attribute aspects 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:

$$\beta \,\text{vol}(\mathcal{E}) = \text{vol}(\mathcal{E}_K), \tag{1}$$

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

$$w(e) = \beta \,\text{vol}(\mathcal{E}) \Big/ \sum_{e \in \mathcal{E}_K} \sum_{v \in e} \gamma(v, e), \forall e \in \mathcal{E}_K. \tag{2}$$

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 $\mathbf{H} = \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_K \end{bmatrix}$, 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 1. Lines 1-4 generate n attribute-based hyperedges $e_i = \text{KNN}(v_i) \cup \{v_i\}$, 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. (2), with $\mathbf{1}_m \mathbf{H}_0 \mathbf{1}_n$ representing $\text{vol}(\mathcal{E})$. 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,

Algorithm 1: ExtendHG

Input: Attributed hypergraph $H = \{\mathcal{V}, \mathcal{E}, \mathbf{X}\}$, parameter K

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

leveraging efficient KNN queries (*e.g.,* [5]), 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 1 achieves log-linear time complexity and linear space complexity for generating $\mathcal{H}$.

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 [39] effectively capture node significance but are limited to pairwise interactions, not applicable to hypergraphs. Other hypergraph definitions [1, 27] 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 [4], 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

$$p(u, v) = \sum_{e \in \mathcal{E}_X} \frac{w(e)\gamma(u,e)}{d(u)} \frac{\gamma(v,e)}{\delta(e)}, \tag{3}$$

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

$$\mathbf{P} = \mathbf{D}_v^{-1} \mathbf{H}^\top \mathbf{W} \mathbf{D}_e^{-1} \mathbf{H}. \tag{4}$$

First, to capture the multi-hop connectivity between nodes, we use the random walk with restart (RWR) model. 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). 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 (5), where $\mathbf{\Pi}^{(t)}[i, j]$ is $\pi^{(t)}(v_i, v_j)$.

$$\mathbf{\Pi}^{(0)} = \mathbf{I}_n, \ \mathbf{\Pi}^{(t+1)} = \alpha \mathbf{I}_n + (1 - \alpha)\mathbf{\Pi}^{(t)} \mathbf{P}, \tag{5}$$

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

$$\mathbf{\Pi}^{(t)} = \textstyle\sum_{i=0}^{t-1} \alpha(1 - \alpha)^i \mathbf{P}^i + (1 - \alpha)^t \mathbf{P}^t. \tag{6}$$

Let $\mathbf{\Pi}$ represent $\mathbf{\Pi}^{(t=\infty)}$ with infinite steps to converge. As one may note, Equations (5) and (6) have similar forms to those in simple graphs. However, our formulation is based on a derivation tailored to 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 into account 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 $\mathbf{p}_s = \mathbf{p}_s \mathbf{P}$ [45]. 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 $p_s(v) = d(v)/\text{vol}(\mathcal{H})$.

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

$$\psi(u, v) = \text{tlog} \frac{\pi(u,v)}{p_s(v)}, \tag{7}$$

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, $\text{tlog}\, x = \log(\max(x, 1))$, is applied to stabilize the ratio against small $p_s(v)$ in large hypergraphs.

By Lemma 1 (proof in Appendix A), 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 1. *For any nodes* $v_i, v_j \in \mathcal{V}$*, we have* $\frac{\pi(v_i,v_j)}{p_s(v_j)} = \frac{\pi(v_j,v_i)}{p_s(v_i)}$.

With Eq. (7), we can express the HMS-N matrix for all $n \times n$ node pairs in $\mathcal{H}$ as

$$\mathbf{\Psi} = \text{tlog}^\circ\left(\text{vol}(\mathcal{H})\mathbf{\Pi}\mathbf{D}_v^{-1}\right), \tag{8}$$

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 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. Then, the node embedding problem of solving $\mathbf{Z}_\mathcal{V}$ is formulated as follows, where $\|\cdot\|_F$ is the Frobenius norm.

$$\mathbf{Z}_\mathcal{V} = \text{argmin}_{\mathbf{Z} \in \mathbb{R}^{n \times k}} \|\mathbf{\Psi} - \mathbf{Z}\mathbf{Z}^\top\|_F^2. \tag{9}$$

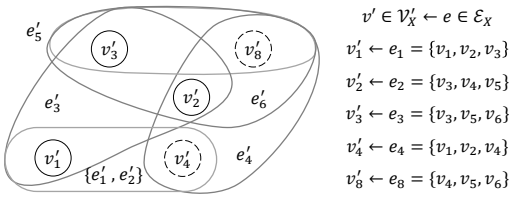


Figure 3: Dual hypergraph $\mathcal{H}'$.

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 $\mathcal{H}' = \{\mathcal{V}'_X, \mathcal{E}'\}$, the new node set $\mathcal{V}'_X$ includes m+n nodes and the new hyperedge set $\mathcal{E}'$ has n hyperedges. Here, each node $v'_i \in \mathcal{V}'_X$ represents the hyperedge $e_i \in \mathcal{E}_X$ of $\mathcal{H}$. Conversely, 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 3 illustrates the dual hypergraph $\mathcal{H}'$ derived from the original $\mathcal{H}$ in Figure 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 $\mathbf{H}' = \mathbf{H}^\top\mathbf{W}$, 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

$$p'(e_i, e_j) = p(v'_i, v'_j) = \textstyle\sum_{v\in\mathcal{V}} \frac{\gamma(v,e_i)}{\delta(e_i)w(e_i)} \frac{\gamma(v,e_j)}{d(v)}, \tag{10}$$

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

$$\mathbf{P}' = (\mathbf{D}_e\mathbf{W})^{-1}(\mathbf{H}^\top\mathbf{W})^\top\mathbf{D}_v^{-1}\mathbf{H}^\top\mathbf{W} = \mathbf{D}_e^{-1}\mathbf{H}\mathbf{D}_v^{-1}\mathbf{H}^\top\mathbf{W}. \tag{11}$$

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. (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 reaches 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 $\mathcal{H}'$, $p_s(v'_j)$, calculated as :

$$p'_s(e_j) = p_s(v'_j) = d(v'_j)/\mathrm{vol}(\mathcal{H}') = \delta(e_j)w(e_j)/\mathrm{vol}(\mathcal{H}). \tag{12}$$

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

$$\psi'(e_i, e_j) = \mathrm{tlog}\,\frac{\pi'(e_i,e_j)}{p'_s(e_j)}. \tag{13}$$

The corresponding HMS-E matrix for all hyperedge pairs is

$$\mathbf{\Psi}' = \mathrm{tlog}^\circ\left(\mathrm{vol}(\mathcal{H})\mathbf{\Pi}'\mathbf{D}_e^{-1}\mathbf{W}^{-1}\right). \tag{14}$$

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

$$\mathbf{Z}_\mathcal{E} = \mathrm{argmin}_{\mathbf{Z}\in\mathbb{R}^{m\times k}} \|\mathbf{\Psi}'_\mathcal{E} - \mathbf{Z}\mathbf{Z}^\top\|_F^2. \tag{15}$$

3.4 A Base Method

In this section, we introduce a basic method to directly address the node and hyperedge embedding objectives in Equations (9) and (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, which achieves comparably high embedding quality with significantly improved efficiency.

For node embeddings, the main idea is to factorize the HMS-N matrix $\mathbf{\Psi}$. Recall that $\mathbf{\Psi}$ in Eq. (8) relies on $\mathbf{\Pi}^{(t=\infty)}$ in Eq. (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. (6). Accordingly, we can derive the approximate HMS-N matrix $\mathbf{\Psi}_T$ by replacing $\mathbf{\Pi}$ with $\mathbf{\Pi}^{(T)}$ in Eq. (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 $\mathbf{\Psi}_T = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$, 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 $\mathbf{Z}_\mathcal{V}\mathbf{Z}_\mathcal{V}^\top$ approximates $\mathbf{\Psi}$ in Eq. (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

$$\mathbf{Z}_\mathcal{V} = \mathbf{Q}_k\mathbf{\Lambda}_\Psi^{1/2}, \text{ where } \mathbf{Q}_k \in \mathbb{R}^{n\times k}, \ \mathbf{\Lambda}_\Psi \in \mathbb{R}^{k\times k}. \tag{16}$$

Similarly, to derive the hyperedge embeddings $\mathbf{Z}_\mathcal{E}$ in Eq. (15), the base method first gets $\mathbf{\Pi}'^{(T)}$ with at most T steps, and 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.

Algorithm 2: 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, 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)
3 **for** $t \leftarrow 1, \ldots, T$ **do**
4 $\quad \mathbf{\Pi}^{(t)} \leftarrow \alpha\mathbf{I}_n + (1-\alpha)\mathbf{\Pi}^{(t-1)}\mathbf{P}$; // Eq. (5)
5 $\mathbf{\Psi}_T \leftarrow \text{tlog}^\circ\left(\text{vol}(\mathcal{H})\mathbf{\Pi}^{(T)}\mathbf{D}_v^{-1}\right)$;
6 $\mathbf{\Lambda}_\Psi, \mathbf{Q}_k \leftarrow \text{eigen}(\mathbf{\Psi}_T, k)$;
7 $\mathbf{Z}_\mathcal{V} \leftarrow \mathbf{Q}_k\mathbf{\Lambda}_\Psi^{1/2}$; // Eq. (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** $t \leftarrow 1, \ldots, T$ **do**
10 $\quad \mathbf{\Pi}'^{(t)} \leftarrow \alpha\mathbf{I}_{m+n} + (1-\alpha)\mathbf{\Pi}'^{(t-1)}\mathbf{P}'$;
11 $\mathbf{\Psi}'_T \leftarrow \text{tlog}^\circ\left(\text{vol}(\mathcal{H})\mathbf{\Pi}'^{(T)}\mathbf{D}_e^{-1}\mathbf{W}^{-1}\right)$;
12 $\mathbf{\Psi}'_\mathcal{E} \leftarrow \mathbf{\Psi}'_T[1:m+1, 1:m+1]$;
13 $\mathbf{\Lambda}'_\Psi, \mathbf{Q}'_k \leftarrow \text{eigen}(\mathbf{\Psi}'_\mathcal{E}, k)$;
14 $\mathbf{Z}_\mathcal{E} \leftarrow \mathbf{Q}'_k\mathbf{\Lambda}'^{1/2}_\Psi$;
15 **return** $\mathbf{Z}_\mathcal{V}, \mathbf{Z}_\mathcal{E}$;

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 2. After constructing the 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, but falls short in scalability. To analyze, Lines 3-4 and Lines 9-10 dominate its time complexity, with time $O(n^2\overline{d}^2)$ and $O((m+n)^2\overline{d}^2)$, 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 $O((m+n)^2\overline{d}^2)$, 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 $\text{tlog}^\circ(\cdot)$ 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 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.

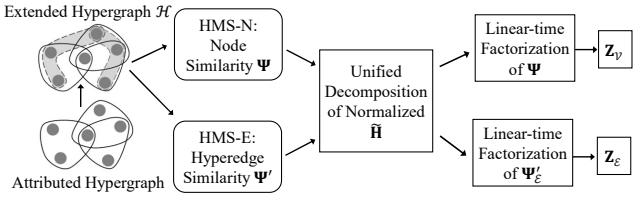


Figure 4: Overview of the SAHE algorithm.

To solve these difficulties, we develop SAHE, an efficient method to produce high-quality AHNEE results, with a complete pipeline outlined by Figure 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 4.1). 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 4.2). In Section 4.3, we present the algorithmic details.

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. (6) and Eq. (8), we need to obtain $\mathbf{\Pi}^{(T)}\mathbf{D}_v^{-1}$ first,

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

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) as follows, and obviously $\mathbf{P}^i\mathbf{D}_v^{-1}$ is symmetric.

$$\mathbf{P}^i\mathbf{D}_v^{-1} = \mathbf{D}_v^{-1/2}\left(\widetilde{\mathbf{H}}^\top\widetilde{\mathbf{H}}\right)^i\mathbf{D}_v^{-1/2}, \tag{17}$$

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 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

$$\mathbf{P}'^i\mathbf{D}_e^{-1}\mathbf{W}^{-1} = \mathbf{D}_e^{-1/2}\mathbf{W}^{-1/2}\left(\widetilde{\mathbf{H}}\widetilde{\mathbf{H}}^\top\right)^i\mathbf{D}_e^{-1/2}\mathbf{W}^{-1/2}. \tag{18}$$

Importantly, observe that in Eq. (17) and Eq. (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. (17) is formulated as

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

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 $\mathbf{\Psi}_T$ can be written as

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

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

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

This can be reformulated as

$$\mathbf{\Psi}_T = \text{tlog}^\circ\left(\mathbf{F}^\top\mathbf{F}\right), \text{where } \mathbf{F} = \sqrt{\text{vol}(\mathcal{H})}\mathbf{D}_v^{-1/2}\mathbf{V}\widehat{\Sigma}^{1/2}. \quad (19)$$

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

$$\mathbf{\Psi}'_T = \text{tlog}^\circ\left(\mathbf{F}'^\top\mathbf{F}'\right), \text{where } \mathbf{F}' = \sqrt{\text{vol}(\mathcal{H})}\mathbf{D}_e^{-1/2}\mathbf{W}^{-1/2}\mathbf{U}\widehat{\Sigma}^{1/2}.$$

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

4.2 HMS-N and HMS-E Approximations

Despite the reformulation, two efficiency issues still remain. First, the reduced SVD of $\widetilde{\mathbf{H}}$ has a prohibitive $O\left(n(m+n)\right)$ time complexity. To improve scalability, we opt for the truncated SVD $\widetilde{\mathbf{H}} \approx \mathbf{U}_r\Sigma_r\mathbf{V}_r^\top$, which only keeps the r largest singular values and the corresponding singular vectors. In this SVD, $\Sigma_r = \text{diag}(\sigma_1, \ldots, \sigma_r)$ 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 Σ, $\mathbf{U}$ and $\mathbf{V}$ with the truncated SVD results, we can derive $\widehat{\Sigma}_r$, $\mathbf{F}_r$ and $\mathbf{F}'_r$, providing rank-r approximations for the node and hyperedge similarity matrices, where the error is bounded in Theorem 2, with proof in Appendix A.

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

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

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

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 $\text{tlog}^\circ(\cdot)$ function. To solve this issue, we employ the polynomial tensor sketch (PTS) technique [11] to approximate $\text{tlog}^\circ(\mathbf{F}_r\mathbf{F}_r^\top)$ 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 using polynomial degree τ and sketch dimension b, leveraging count-sketch matrices and recursive calculations via the fast Fourier transform. Then, we estimate polynomial coefficients through regression with sample size c to obtain the full approximation. 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 $\text{tlog}^\circ(\cdot)$ via regression. Moreover, the approximation error of PTS is bounded by Lemma 3 for our approximation based on the theory in [11].

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

Although $\Gamma = \mathbf{Y}\mathbf{\Theta}\mathbf{Y}^\top$ resembles the eigendecomposition of Γ, 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 Γ by iteratively applying the linear operator $\mathcal{L}(\mathbf{v}) = \Gamma\mathbf{v} = \mathbf{Y}\left(\mathbf{\Theta}\left(\mathbf{Y}^\top\mathbf{v}\right)\right)$, which multiplies a vector $\mathbf{v}$ by Γ with complexity linear to n. Consequently, the matrix $\mathbf{\Lambda}_\Gamma$ contains the k largest-magnitude eigenvalues of Γ, and $\mathbf{Q}_\Gamma$ comprises their corresponding eigenvectors as columns, yielding a factorization: $\mathbf{Q}_\Gamma\mathbf{\Lambda}_\Gamma\mathbf{Q}_\Gamma^\top$. Finally, we get the node embeddings as

$$\mathbf{Z}_\mathcal{V} = \mathbf{Q}_\Gamma\mathbf{\Lambda}_\Gamma^{1/2}. \quad (20)$$

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

$$\mathbf{Z}_\mathcal{E} = \mathbf{Q}'_\Gamma\mathbf{\Lambda}'^{1/2}_\Gamma. \quad (21)$$

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 3.

Algorithm. After constructing the attribute-extended hypergraph $\mathcal{H}$ at Line 1 by Algorithm 1 (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 obtain the normalized hypergraph incidence matrix $\widetilde{\mathbf{H}}$ and decompose it into $\mathbf{U}_r, \Sigma_r$, and $\mathbf{V}_r$ via the rank-r TruncatedSVD (Lines 2-3). We calculate $\widehat{\Sigma}_r$ from the r largest singular values of $\widetilde{\mathbf{H}}$ in Lines 4-6. Then we derive the embeddings for nodes and hyperedges.

For node embeddings (Lines 7-11), we first derive $\mathbf{F}_r$ by its definition in Theorem 2, and the node similarity matrix becomes $\text{tlog}^\circ\left(\mathbf{F}_r\mathbf{F}_r^\top\right)$. To compute this $\text{tlog}^\circ\left(\mathbf{F}_r\mathbf{F}_r^\top\right)$ function efficiently, we derive its polynomial tensor sketches $\mathbf{Y}$ and $\mathbf{\Theta}$. Finally, the node embeddings $\mathbf{Z}_\mathcal{V} = \mathbf{Q}_\Gamma\mathbf{\Lambda}_\Gamma^{1/2}$ are obtained by factorizing $\Gamma = \mathbf{Y}\mathbf{\Theta}\mathbf{Y}^\top$ into $\mathbf{Q}_\Gamma\mathbf{\Lambda}_\Gamma\mathbf{Q}_\Gamma^\top$ via the Lanczos technique [19].

Algorithm 3: 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 $\mathbf{H}, \mathbf{D}_v, \mathbf{D}_e, \mathbf{W} \leftarrow \texttt{ExtendHG}(\mathbf{H}_0, \mathbf{X}, K)$;
2 $\widetilde{\mathbf{H}} \leftarrow \mathbf{W}^{1/2}\mathbf{D}_e^{-1/2}\mathbf{H}\mathbf{D}_v^{-1/2}$; // Eq. (17)
3 $\mathbf{U}_r, \mathbf{\Sigma}_r, \mathbf{V}_r \leftarrow \texttt{TruncatedSVD}\left(\widetilde{\mathbf{H}}, r\right)$;
4 $\widehat{\mathbf{\Sigma}}_r \leftarrow \mathbf{I}_r$;
5 **for** $i \leftarrow 1, \ldots, T$ **do**
6 $\quad \widehat{\mathbf{\Sigma}}_r \leftarrow \alpha\mathbf{I}_r + (1-\alpha)\mathbf{\Sigma}_r^2\widehat{\mathbf{\Sigma}}_r$;
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 2
8 $\mathbf{Y}, \mathbf{\Theta} \leftarrow \texttt{PTS}(\mathbf{F}_r, \mathrm{tlog}, \tau, b, c)$;
9 Linear operator $\mathcal{L}(\mathbf{v}) = \mathbf{Y}\left(\mathbf{\Theta}\left(\mathbf{Y}^\top\mathbf{v}\right)\right)$;
10 $\mathbf{\Lambda}_\Gamma, \mathbf{Q}_\Gamma \leftarrow \texttt{Lanczos}(\mathcal{L}, k)$; // eigen $\left(\mathbf{Y}\mathbf{\Theta}\mathbf{Y}^\top, k\right)$
11 $\mathbf{Z}_\mathcal{V} \leftarrow \mathbf{Q}_\Gamma\mathbf{\Lambda}_\Gamma^{1/2}$; // Eq. (20)
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 $\mathbf{Y}', \mathbf{\Theta}' \leftarrow \texttt{PTS}\left(\mathbf{F}'_r[1:m+1,:], \mathrm{tlog}, \tau, b, c\right)$;
14 Linear operator $\mathcal{L}'(\mathbf{v}) = \mathbf{Y}'\left(\mathbf{\Theta}'\left(\mathbf{Y}'^\top\mathbf{v}\right)\right)$;
15 $\mathbf{\Lambda}'_\Gamma, \mathbf{Q}'_\Gamma \leftarrow \texttt{Lanczos}\left(\mathcal{L}', k\right)$; // eigen $\left(\mathbf{Y}'\mathbf{\Theta}'\mathbf{Y}'^\top, k\right)$
16 $\mathbf{Z}_\mathcal{E} \leftarrow \mathbf{Q}'_\Gamma\mathbf{\Lambda}'^{1/2}_\Gamma$; // Eq. (21)
17 **return** $\mathbf{Z}_\mathcal{V}, \mathbf{Z}_\mathcal{E}$;

Following a similar process, we can derive the hyperedge embeddings $\mathbf{Z}_\mathcal{E} = \mathbf{Q}'_\Gamma\mathbf{\Lambda}'^{1/2}_\Gamma$ (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, invoking 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 $n\overline{d} + nK$ nonzero entries. The TruncatedSVD technique in Line 3 involves a bounded number of matrix-vector multiplications on $\widetilde{\mathbf{H}}$, and hence incurs $O(n\overline{d}+nK)$ 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 node embeddings, we compute $\mathbf{F}_r$ in Line 7, which takes $O(nr)$ time. Next, recall from Section 4.2 that the approximation via the PTS method in Line 8 finishes in linear time $O(n)$. Regarding the Lanczos method in Lines 9-10, the linear operator $\mathcal{L}(\cdot)$ executes in linear time and is applied a constant $O(1)$ number of times. Hence, its computation remains linear in 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 $O(n \log n + n\overline{d} + nq + m)$, or simply $O(n \log n + m)$ as q and $\overline{d}$ can be considered constant. Moreover, the memory overhead of SAHE is $O(n\overline{d} + nq + m)$, 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 achieved by SAHE is well worth the approximation trade-offs.

Table 2: Dataset Statistics.

Dataset	n	m	$\overline{d}$	$\overline{\delta}$	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

5 EXPERIMENTS

After providing the experimental settings in Section 5.1, we report the performance of node embedding on node classification task in Section 5.2 and on hyperedge link prediction task in Section 5.3, and the performance of hyperedge embedding on hyperedge classification task in Section 5.4. The efficiency results and experimental analysis are in Section 5.5 and Section 5.6.

5.1 Experimental Setup

Datasets. Table 2 summarizes the statistics of attributed hypergraphs used in our experiments, including the number of nodes (n) and hyperedges (m), the average node degree ($\overline{d}$), the average hyperedge size ($\overline{\delta}$), the dimension of node attributes (q), and the number of ground-truth class labels (ℓ). DBLP-CA, Cora-CA, Cora-CC, Citeseer, and DBLP are benchmark datasets in [36]. Mushroom and 20News are from [3], and Recipe is from [20]. Amazon and MAG-PM are million-scale from [21]. 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 [14], and three attributed graph embedding approaches

Table 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

(*i.e.,* PANE [41], AnECI [23], and CONN [28]), which are applied to reduced graphs derived from the clique expansion of the hypergraph. Also, we consider two bipartite graph embedding techniques AnchorGNN [32] and BiANE [15] that are applied to a bipartite graph where hyperedges are treated as a distinct set of nodes separate from the original nodes. Finally, we include three self-supervised learning baselines (VilLain [18], TriCL [17], and HypeBoy [16]), with TriCL and HypeBoy targeted for attributed hypergraph embedding, and matrix factorization approaches on the general graph, NetMF [26] and LightNE [25]. 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 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 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 3-5.

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 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 3.4, SAHE, developed in Section 4, is outperformed slightly on small datasets but excels on large ones where Base is inefficient. This highlights the effectiveness 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 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 3 and algorithm designs in Section 4.2.

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 [18, 24]. For each dataset, we divide hyperedges into training and test sets, using an 80%/20% split for smaller 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

Table 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

Table 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

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 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.

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 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. 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 3.3 and 4.

5.5 Embedding Efficiency

Figure 5 reports the time of all methods to generate node and hyperedge embeddings across the ten 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 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 5, often being faster by orders of magnitude. Taking the Citeseer dataset as example, Figure 5d shows that SAHE is 422.5× faster than BiANE (3rd place in node classification), 15.3× faster

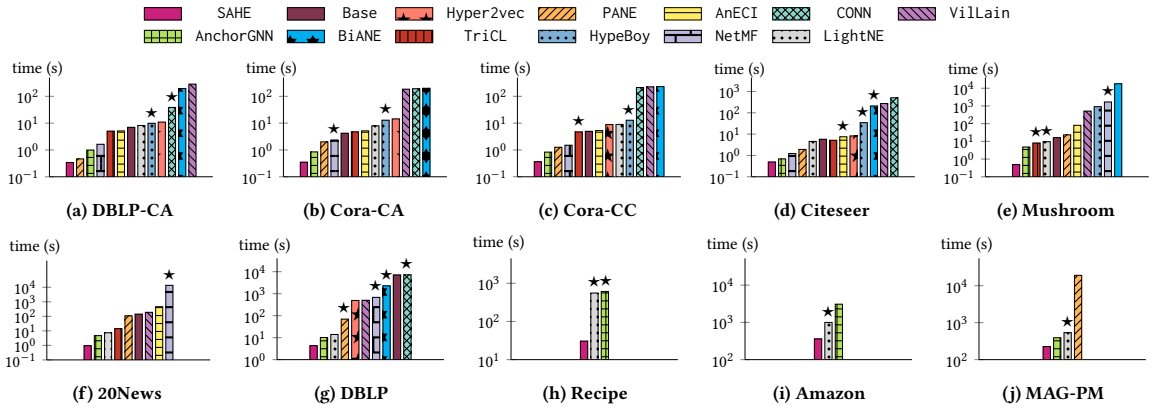


Figure 5: Running time of generating node and hyperedge embeddings (★ marks the best competitors in Tables 3, 4, 5).

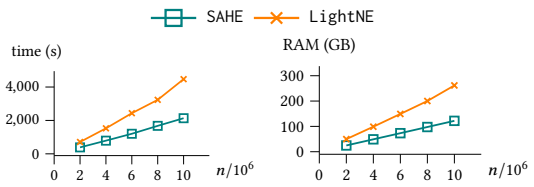


Figure 6: Scalability Test.

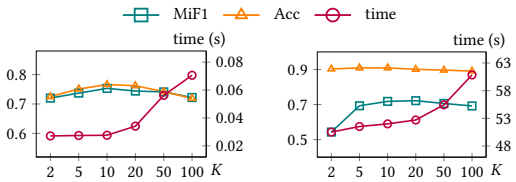


Figure 7: Varying K.

Table 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

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 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 3. These results underscore the combination of high-quality embeddings and excellent efficiency achieved by SAHE.

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 [9], and each node is assigned 100 random binary attributes. Figure 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 4.3 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 6 quantifies this loss by reporting the mean absolute error (MAE) 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 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.

Varying β. Parameter β balances attribute-based hyperedges $\mathcal{E}_K$ and original hyperedges $\mathcal{E}$ in $\mathcal{H}$. Figure 8 shows that as β increases from 0.1 to 1, micro-F1 for node embedding generally increases, and stabilizes beyond 1.0 on most datasets, but declines 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, and then remains stable. 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 4.2. We vary r for node classification (MiF1) and hyperedge link prediction (Acc), with results in Figure 9. Increasing r from 16 to 32 generally improves or stabilizes both metrics, except for Mushroom, where

Table 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**

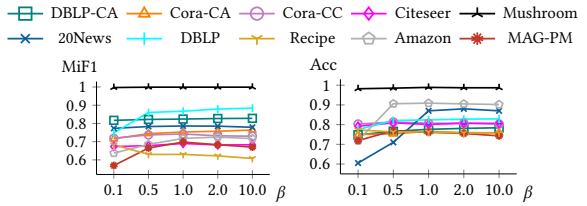


Figure 8: Varying β.

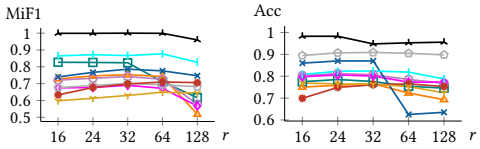


Figure 9: Varying r.

Table 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**

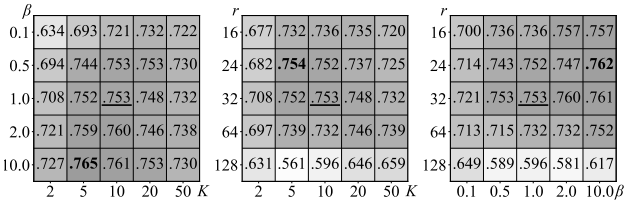


Figure 10: Heatmaps between K, β, and r on Cora-CA for node classification. Darker shades indicate higher MiF1. Underlined results are reported in Table 3, while the optimal parameter combinations are in bold.

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 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 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 evaluate 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 7-8. 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 9-11 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 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 12(a) and (b) show MiF1 for node classification and running time as τ varies from 1 to 9, while Figure 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.

6 RELATED WORK

Existing methods struggle to support attributed hypergraphs natively while scaling for massive data, particularly for the AHNEE problem that embeds both nodes and hyperedges. Early hypergraph embedding efforts, like [45], use the Laplacian matrix spectrum for node embeddings, focusing on clustering but neglecting attributes and long-range connectivity. Methods extending node2vec [10] to

Table 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 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 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**

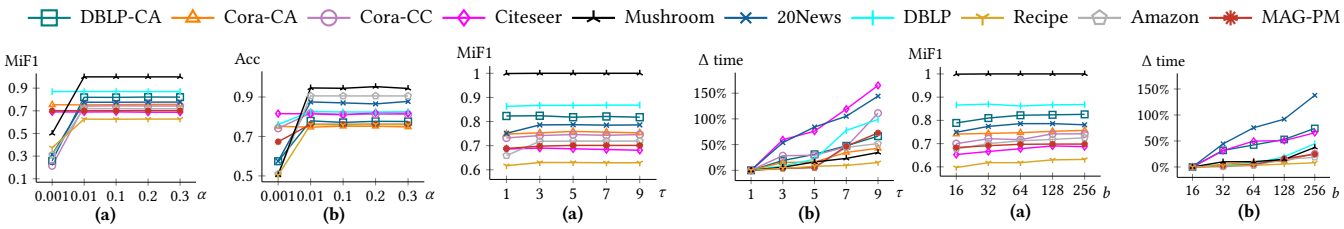


Figure 11: Varying α. **Figure 12: Varying τ.** **Figure 13: Varying b.**

hypergraphs, such as Hyper2vec [14] and its dual-enhanced version [13], capture long-range relationships via random walks, yet omit attributes and hyperedge embeddings. Another approach [44] models hyperedges as multi-linear products of node embeddings, but lacks attribute consideration and scalability on large datasets.

The emergence of hypergraph neural networks also enables a multitude of recent approaches. TriCL [17] uses contrastive learning to embed nodes, HypeBoy [16] masks attributes and designs a hyperedge-filling task for node embedding, and [6] 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. VilLain [18] formulates the self-supervision as a label propagation process while ignoring node attributes and incurring high training costs. There are specialized methods to learn node representations for certain hypergraphs, like [38] for location-based networks, [33] for recommendations, [35] for trust relations, and [22] for 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. With each hyperedge converted to a fully connected subgraph via clique-expansion, hypergraphs can be processed by graph embedding methods (NetMF [26], STRAP [43], LightNE [25], etc.) or attributed graph embedding method [42] based on matrix factorization. Star expansion of hypergraphs results in dense edges between two sets of nodes, enabling the adoption of bipartite graph embedding methods (BiANE [15], AnchorGNN [32]). 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.

7 CONCLUSION

This paper proposes the SAHE algorithm to tackle the AHNEE problem, by efficiently generating node and hyperedge embeddings that preserve the higher-order connectivity among nodes and collective similarities among hyperedges in attributed hypergraphs. By formulating multi-hop similarity measures and employing optimized decomposition techniques, SAHE achieves log-linear time complexity, outperforming 11 baselines across 10 real-world datasets. Its scalability and effectiveness make it well-suited for practical applications. Moving forward, we aim to enhance SAHE with a dynamic

embedding algorithm using incremental decomposition for evolving hypergraphs and a GPU implementation to boost efficiency.

ACKNOWLEDGMENTS

This work is supported by grants from the Research Grants Council of Hong Kong Special Administrative Region, China (No. PolyU 25201221, No. PolyU 15205224), and NSFC No. 62202404. This work is supported by Smart Cities Research Institute (SCRI) P0051036-P0050643, and grant P0048213 from Tencent Technology Co., Ltd.

A PROOFS

Proof of Lemma 1. The random walk on $\mathcal{H}$ has stationary probability $p_s(v_i) = d(v_i)/\mathrm{vol}(\mathcal{H})$ [45]. Thus, the matrix with $\mathbf{p}_s$ as diagonal is $\mathbf{D}_v/\mathrm{vol}(\mathcal{H})$. Two sides of the lemma are written in matrix form as $\mathrm{vol}(\mathcal{H})\mathbf{\Pi}\mathbf{D}_v^{-1}$ and $(\mathrm{vol}(\mathcal{H})\mathbf{\Pi}\mathbf{D}_v^{-1})^\top$. Then we get $\mathbf{D}_v^{-1}\mathbf{\Pi}^\top = \mathbf{D}_v^{-1}\sum_{i=0}^{\infty}\alpha(1-\alpha)^i(\mathbf{P}^i)^\top = \sum_{i=0}^{\infty}\alpha(1-\alpha)^i\mathbf{D}_v^{-1}\left(\mathbf{H}^\top\mathbf{W}\mathbf{D}_e^{-1}\mathbf{H}\mathbf{D}_v^{-1}\right)^i = \sum_{i=0}^{\infty}\alpha(1-\alpha)^i\mathbf{P}^i\mathbf{D}_v^{-1} = \mathbf{\Pi}\mathbf{D}_v^{-1} = (\mathbf{D}_v^{-1}\mathbf{\Pi}^\top)^\top$. □

Proof of Theorem 2. 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 written 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

$$\begin{aligned}|\mathbf{A}[i,j]| &= \left|\mathrm{tlog}\left(\tilde{\mathbf{F}}_r[i,j]\right) - \mathrm{tlog}\left(\tilde{\mathbf{F}}[i,j]\right)\right| \\ &= \left|\log\left(\max\left\{\tilde{\mathbf{F}}_r[i,j],1\right\}\right) - \log\left(\max\left\{\tilde{\mathbf{F}}[i,j],1\right\}\right)\right| \\ &\le \left|\max\left\{\tilde{\mathbf{F}}_r[i,j],1\right\} - \max\left\{\tilde{\mathbf{F}}[i,j],1\right\}\right| \le \left|\tilde{\mathbf{F}}_r[i,j] - \tilde{\mathbf{F}}[i,j]\right|.\end{aligned} \tag{22}$$

The second equality is due to $\mathrm{tlog}^\circ(\cdot)$. The first inequality is because $|\log x - \log y| \le |x-y|$ for any $x, y \ge 1$. The second inequality is because $|\max\{x,1\} - \max\{y,1\}| \le |x-y|$ 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

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

where the first inequality is due to Ineq. (22). The second inequality is due to the property of the 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 $\sum_i a_i^2 \le (\sum_i a_i)^2$ where $a_i \ge 0$. Deriving the upper bound in the second inequality of Theorem 2 is similar and thus omitted. □

REFERENCES

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

[2] Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon Kleinberg. 2018. Simplicial Closure and Higher-Order Link Prediction. *PNAS* 115, 48 (2018), E11221–E11230.

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

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

[5] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

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

[7] 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. 2021. Hypergraph Models of Biological Networks to Identify Genes Critical to Pathogenic Viral Response. *BMC Bioinformatics* 22, 1 (May 2021), 287.

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

[9] Yue Gao, Yifan Feng, Shuyi Ji, and Rongrong Ji. 2023. HGNN+: General Hypergraph Neural Networks. https://github.com/iMoonLab/DeepHypergraph. *TPAMI* 45, 3 (March 2023), 3181–3199.

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

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

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

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

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

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

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

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

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

[19] R. B. Lehoucq and D. C. Sorensen. 1996. Deflation Techniques for an Implicitly Restarted Arnoldi Iteration. *SIAM J. Matrix Anal. Appl.* 17, 4 (1996), 789–821.

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

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

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

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

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

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

[26] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and Node2vec. In *WSDM*. 459–467.

[27] Yuuki Takai, Atsushi Miyauchi, Masahiro Ikeda, and Yuichi Yoshida. 2020. Hypergraph Clustering Based on PageRank. *arXiv:2006.08302 [cs, math]* (June 2020). arXiv:2006.08302

[28] Qiaoyu Tan, Xin Zhang, Xiao Huang, Hao Chen, Jundong Li, and Xia Hu. 2023. Collaborative graph neural networks for attributed network embedding. *TKDE* (2023).

[29] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan Oseledets, and Emmanuel Müller. 2021. FREDE: Anytime Graph Embeddings. *VLDB* 14, 6 (Feb. 2021), 1102–1110.

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

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

[32] Xueyi Wu, Yuanyuan Xu, Wenjie Zhang, and Ying Zhang. 2023. Billion-Scale Bipartite Graph Embedding: A Global-Local Induced Approach. *VLDB* 17, 2 (Oct. 2023), 175–183.

[33] Lianghao Xia, Chao Huang, and Chuxu Zhang. 2022. Self-Supervised Hypergraph Transformer for Recommender Systems. In *KDD*. 2100–2109.

[34] Mengjia Xu. 2021. Understanding Graph Embedding Methods and Their Applications. *SIAM Rev.* 63, 4 (Jan. 2021), 825–853.

[35] Rongwei Xu, Guanfeng Liu, Yan Wang, Xuyun Zhang, Kai Zheng, and Xiaofang Zhou. 2024. Adaptive Hypergraph Network for Trust Prediction. In *ICDE*. 2986–2999.

[36] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. 2019. Hypergcn: A new method for training graph convolutional networks on hypergraphs. *NeurIPS* 32 (2019).

[37] Yuguang Yan, Yuanlin Chen, Shibo Wang, Hanrui Wu, and Ruichu Cai. 2024. Hypergraph Joint Representation Learning for Hypervertices and Hyperedges via Cross Expansion. *AAAI* 38, 8 (March 2024), 9232–9240.

[38] Dingqi Yang, Bingqing Qu, Jie Yang, and Philippe Cudré-Mauroux. 2022. LBSN2Vec++: Heterogeneous Hypergraph Embedding for Location-Based Social Networks. *TKDE* 34, 4 (April 2022), 1843–1855.

[39] Mingji Yang, Hanzhi Wang, Zhewei Wei, Sibo Wang, and Ji-Rong Wen. 2024. Efficient Algorithms for Personalized PageRank Computation: A Survey. *TKDE* 36, 9 (Sept. 2024), 4582–4602.

[40] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S. Bhowmick. 2020. Homogeneous Network Embedding for Massive Graphs via Reweighted Personalized PageRank. *VLDB* 13, 5 (Jan. 2020), 670–683.

[41] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Sourav S. Bhowmick, and Juncheng Liu. 2023. PANE: Scalable and Effective Attributed Network Embedding. *VLDBJ* 32, 6 (Nov. 2023), 1237–1262.

[42] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Juncheng Liu, and Sourav S. Bhowmick. 2020. Scaling Attributed Network Embedding to Massive Graphs. *VLDB* 14, 1 (Sept. 2020), 37–49.

[43] Yuan Yin and Zhewei Wei. 2019. Scalable Graph Embeddings via Sparse Transpose Proximities. In *KDD (KDD '19)*. 1429–1437.

[44] Chia-An Yu, Ching-Lun Tai, Tak-Shing Chan, and Yi-Hsuan Yang. 2018. Modeling Multi-way Relations with Hypergraph Embedding. In *CIKM (CIKM '18)*. 1707–1710.

[45] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. 2007. Learning with Hypergraphs: Clustering, Classification, and Embedding. In *NeurIPS*, Vol. 19.

[46] Jingya Zhou, Ling Liu, Wenqi Wei, and Jianxi Fan. 2022. Network Representation Learning: From Preprocessing, Feature Extraction to Node Embedding. *ACM Comput. Surv.* 55, 2 (Jan. 2022), 38:1–38:35.