Random Walk with Restart on Hypergraphs: Fast Computation and an Application to Anomaly Detection

Jaewan Chun [1], Geon Lee [1], Kijung Shin [1,*], Jinhong Jung [2,*]

[1] Kim Jaechul Graduate School of AI, KAIST, Seoul, South Korea,
[2] School of Software, Soongsil University, Seoul, South Korea,
jjwpalace@kaist.ac.kr, geonlee0325@kaist.ac.kr, kijungs@kaist.ac.kr, jinhong@ssu.ac.kr

Abstract

Random walk with restart (RWR) is a widely-used measure of node similarity in graphs, and it has proved useful for ranking, community detection, link prediction, anomaly detection, etc. Since RWR is typically required to be computed separately for a larger number of query nodes or even for all nodes, fast computation of it is indispensable. However, for hypergraphs, the fast computation of RWR has been unexplored, despite its great potential. In this paper, we propose ARCHER, a fast computation framework for RWR on hypergraphs. Specifically, we first formally define RWR on hypergraphs, and then we propose two computation methods that compose ARCHER. Since the two methods are complementary (i.e., offering relative advantages on different hypergraphs), we also develop a method for automatic selection between them, which takes a very short time compared to the total running time. Through our extensive experiments on 18 real-world hypergraphs, we demonstrate (a) the speed and space efficiency of ARCHER, (b) the complementary nature of the two computation methods composing ARCHER, (c) the accuracy of its automatic selection method, and (d) its successful application to anomaly detection on hypergraphs.

Keywords: Hypergraph, Random Walk with Restart, Fast Computation, Anomaly Detection

1 Introduction

Given a pair of nodes on a large-scale hypergraph, how can we rapidly and efficiently calculate their proximity based on random walk with restart? How useful are such proximities for data-mining applications?

A hypergraph is a data structure consisting of a set of nodes and a set of hyperedges, and each hyperedge is a set composed of any number of nodes. Note that a hypergraph where each hyperedge joins any number of nodes is a generalization of a (pairwise) graph where each edge always joins two nodes. Due to this increased expressiveness, hypergraphs are widely used to model real-world group relations, including (two or more) researchers who co-author a paper, items purchased together, and tags co-appearing in a post [1, 2, 3, 4].

Random walk, a concept widely used for graphs, naturally applies to hypergraphs. Random walk is a stochastic process that assumes an imaginary surfer moving randomly between nodes in a graph; and for instance, PageRank [5] utilizes this concept to model a random surfer navigating a web graph (i.e., a network of hyperlinks between web pages) and quantifies the importance of web pages based on the stationary distribution of the surfer's visits. A simple hypergraph extension [6] assumes a random surfer that repeats (a) choosing an incident hyperedge with probability proportional to edge weights and (b) choosing an incident node uniformly at random. However, its expressiveness is limited in that such random walk can always be reduced to random walk on an undirected graph, with some choice of weights [7]. [7] proposed a more expressive one that may not be reduced to random walk on any undirected graph. It assumes a random surfer who uses edge-dependent node weights (EDNW) when choosing incident nodes, and due to its expressiveness, it has been employed for clustering [8], product-return prediction [9], object classification [10], anomaly detection [11], etc.

*Co-corresponding authors.

On graphs, the concept of random walk with restart (RWR) [12] has also been widely used. RWR measures the stationary probability distribution of random walk when we assume a random surfer who restarts at a query node with a certain probability, and the distribution is naturally interpreted as the relevance of each node with respect to the query node. Due to its ability to consider multi-faceted relationships between nodes, RWR has been extensively utilized in graph mining applications including personalized ranking [12, 13], anomaly detection [14], subgraph mining [15], graph neural networks [16], and graph augmentation [17, 18]. Since RWR is typically required to be computed separately for a larger number of query nodes or even for all nodes, fast computation of it is indispensable. Therefore, many computation methods have been developed [19, 20], and many of them rely on preprocessing the input graph [15, 21, 22, 23].

However, RWR on hypergraphs has been underexplored, while an extension of RWR to hypergraphs is straightforward with many potential applications. One potential reason is the lack of fast and scalable computation methods of it. For example, the previous work [7] has relied on naive power iteration, which becomes impractical when dealing with a large number of query nodes on large-scale hypergraphs. Since random walk on a hypergraph consists of two-step transitions (i.e., from node to hyperedge, and from hyperedge to node), we cannot directly employ existing fast and scalable computation methods for RWR on a graph whose transition is just from node to node. Similar to graphs, RWR scores on hypergraphs vary across different query nodes, and computing RWR scores for a large number of query nodes is necessary for applications. Thus, dedicated efforts are necessary to develop fast computation methods can offer low costs per query node, even if it involves incurring a one-time preprocessing cost.

In this work, we propose ARCHER (Adaptive RWR Computation on Hypergraphs), a fast and space-efficient framework for computing RWR on real-world hypergraphs. After formally defining RWR on hypergraphs, we develop two computation methods for it based on two different (simplified) representations of hypergraphs. These two computation methods are complementary, and they offer relative advantages on different hypergraphs. Thus, we further propose an automatic selection method that chooses one computation method based on a simple but effective goodness criterion whose computation takes a very short time compared to the total running time.

Through extensive experiments on 18 real-world hypergraphs, we substantiate the speed and space efficiency of ARCHER and its two key driving factors: (a) the complementarity between the two computation methods composing ARCHER and (b) the accuracy of its automatic selection method. In addition, we demonstrate a successful application of RWR on hypergraphs and ARCHER to anomaly detection.

Reproducibility. The source code and datasets used in this paper are available at `https://github.com/jaewan01/ARCHER`.

The rest of the paper is organized as follows. In Section 2, we introduce preliminaries and related work. In Section 3, we describe our approaches for computing RWR on hypergraphs. In Section 4, we introduce an application of RWR on hypergraphs for the purpose of anomaly detection. After sharing experimental results in Section **??**, we present conclusions and future directions in Section 6.

2 Preliminaries and Related Work

In this section, we introduce some preliminaries and related studies on random walks on (hyper-)graphs and their applications.

2.1 Notations

We describe basic notations frequently used in this paper where related symbols are summarized in Table 1.

(Hyper)graph. A *hypergraph* $G_H = (\mathcal{V}, \mathcal{E}, \omega, \gamma)$ consists of a set $\mathcal{V}$ of nodes, a set $\mathcal{E}$ of hyperedges, the weight $\omega(e)$ of hyperedge e, and the weight $\gamma_e(v)$ of node v depending on hyperedge e. Each hyperedge $e \in E$ is represented by a non-empty subset of an arbitrary number of nodes, i.e., $e \in 2^V$. We let $n = |\mathcal{V}|$ and $m = |\mathcal{E}|$ be the numbers of nodes and hyperedges, respectively. Similarly, a *graph* $G = (V, E, w)$ consists of a set V of nodes, a set E of edges, and edge weights w.

Matrix representation. Consider a one-to-one mapping f between $\mathcal{V}$ and $\{1, \cdots, n\}$ and a one-to-one mapping g between $\mathcal{E}$ and $\{1, \cdots, m\}$. For any matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$, we denote its $(f(v), g(e))$-th entry

Table 1: Symbols.

Symbol	Definition
$G_H = (\mathcal{V}, \mathcal{E}, \omega, \gamma)$	hypergraph where $\mathcal{V}$ and $\mathcal{E}$ are sets of nodes and hyperedges, $\omega(e)$ is the weight of hyperedge e, and $\gamma_e(v)$ is the weight of node v depending on hyperedge e
$n = \lvert\mathcal{V}\rvert$	number of nodes
$m = \lvert\mathcal{E}\rvert$	number of hyperedges
$\mathbf{W} \in \mathbb{R}^{n \times m}$	hyperedge-weight matrix of G_H and $\tilde{\mathbf{W}}$ is row-normalized
$\mathbf{R} \in \mathbb{R}^{m \times n}$	node-weight matrix of G_H and $\tilde{\mathbf{R}}$ is row-normalized
$G = (V, E, w)$	graph where V is the node set, E is the edge set, and $w(e)$ is the weight of edge e
$G_{\mathcal{C}}$	clique-expanded graph from G_H
$G_\star$	star-expanded graph from G_H
$E(v)$	hyperedges incident to node v, i.e., $E(v) = \{e \in \mathcal{E} : v \in e\}$
$\bar{d}(v)$	unweighted degree of node v, i.e., $\bar{d}(v) = \lvert E(v)\rvert$
$\lvert e\rvert$	size of hyperedge e
$\gamma(v)$	degree of hyperedge e, i.e., $\gamma(v) = \sum_{v \in e} \gamma_e(v)$
s	query node for RWR
c	restart probability of RWR where $0 < c < 1$
$c_\star$	modified restart probability, i.e, $c_\star = 1 - \sqrt{1-c}$
$\boldsymbol{q} \in \mathbb{R}^n$	RWR query vector w.r.t query node s
$\boldsymbol{r} \in \mathbb{R}^n$	RWR score vector w.r.t. query node s
$\mathbf{H}_{\mathcal{C}} \in \mathbb{R}^{n \times n}$	$\mathbf{H}_{\mathcal{C}} = \mathbf{I}_n - (1-c)\tilde{\mathbf{P}}^\top$ where $\tilde{\mathbf{P}} = \tilde{\mathbf{W}}\tilde{\mathbf{R}}$
$\mathbf{H}_\star \in \mathbb{R}^{N \times N}$	$\mathbf{H}_\star = \mathbf{I}_N - (1-c_\star)\tilde{\mathbf{S}}^\top$ where $\tilde{\mathbf{S}} = \left(\begin{smallmatrix} \mathbf{0} & \tilde{\mathbf{W}} \\ \tilde{\mathbf{R}} & \mathbf{0} \end{smallmatrix}\right)$ and $N = n + m$

$\mathbf{X}_{f(v)g(e)}$ simply by $\mathbf{X}_{ve}$. Similarly, for any matrix $\mathbf{Y} \in \mathbb{R}^{m \times n}$, we let $\mathbf{Y}_{ev}$ denote $\mathbf{Y}_{g(e)f(v)}$, and for any $\mathbf{Z} \in \mathbb{R}^{n \times n}$, we let $\mathbf{Z}_{vv}$ denote $\mathbf{Z}_{f(v)f(v)}$. The matrix $\mathbf{W} \in \mathbb{R}^{n \times m}$ is the *hyperedge-weight matrix* whose entry $\mathbf{W}_{ve} = \omega(e)$ if $v \in e$, and 0 otherwise. The matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$ is the *node-weight matrix* whose entry $\mathbf{R}_{ev} = \gamma_e(v)$ if $v \in e$, and 0 otherwise. In the *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{\lvert V\rvert \times \lvert V\rvert}$ of any graph G, $\mathbf{A}_{uv} = w(e)$ if there is an edge between nodes $u \in V$ and $v \in V$; otherwise, $\mathbf{A}_{uv} = 0$.

Hypergraph expansions. A hypergraph can be converted into graphs using clique- and star-expansion. Clique expansion [24] constructs a graph $G_{\mathcal{C}} = (\mathcal{V}, E_{\mathcal{C}})$ from G_H by replacing each original hyperedge with a clique composed of the nodes in the hyperedge, i.e., $E_{\mathcal{C}} = \{(u, v) | u, v \in e, e \in \mathcal{E}\}$. Notably, the adjacency matrix of $G_{\mathcal{C}}$ has the same sparsity pattern as $\mathbf{P} = \mathbf{WR}$, as illustrated in Figure 1.

Star expansion [25] constructs a graph $G_\star = (V_\star, E_\star)$ by aggregating nodes and hyperedges as a new set of nodes (i.e., $V_\star = \mathcal{V} \cup \mathcal{E}$), and edges are created between each pair of incident node and hyperedge (i.e., $E_\star = \{(v, e) | v \in e, v \in \mathcal{V}, e \in \mathcal{E}\}$). The sparsity pattern of the adjacency matrix of $G_\star$ is the same as $\mathbf{S} = \left(\begin{smallmatrix} \mathbf{0} & \mathbf{W} \\ \mathbf{R} & \mathbf{0} \end{smallmatrix}\right)$, as illustrated in Figure 1.

2.2 Random Walk with Restart on Graphs

We introduce the concept of random walk with restart (RWR) on a graph and existing methods for computing RWR scores.

Concept. Given a graph G and a query node s, random walk with restart (RWR) aims to obtain a vector $\boldsymbol{r}$ of proximities from s to each node on the graph [12]. Specifically, it assumes a random surfer that starts from node s and takes one of the following actions at each step:

- **Action 1) Random walk.** The surfer randomly moves to one of the neighbors from the current node with probability $1 - c$. The probability of selecting each neighbor is proportional to the edge weight between the current node and the neighbor.
- **Action 2) Restart.** The surfer jumps back to the query node s with restart probability c.

The stationary probability of the surfer visiting a node u is denoted by $\mathbf{r}_u$. That is, the RWR score vector $\boldsymbol{r}$ of all nodes w.r.t. s (a.k.a. single-source RWR scores) is the unique solution of the following equation:

$$\boldsymbol{r} = (1-c)\tilde{\mathbf{A}}^\top \boldsymbol{r} + c\boldsymbol{q}, \tag{1}$$

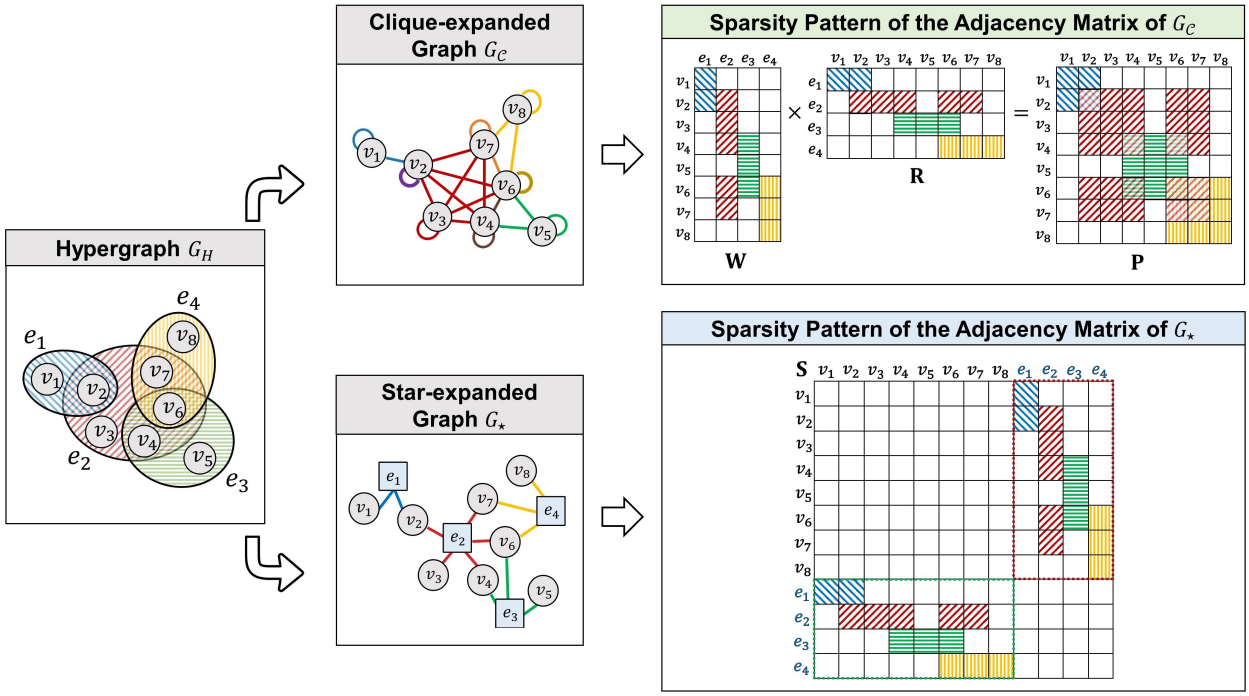


Figure 1: Clique- and star-expansions of an example hypergraph and their sparsity patterns.

where $\tilde{\mathbf{A}}$ is the row-normalized adjacency matrix of G, and c is called *restart probability*. An RWR query is denoted by $\boldsymbol{q} \in \mathbb{R}^n$, which is a unit vector whose s-th entry is 1. The resulting RWR score vector for the query $\boldsymbol{q}$ is denoted by $\boldsymbol{r} \in \mathbb{R}^n$. The choice of the query node s determines a specific RWR query $\boldsymbol{q}$, leading to a distinct RWR score vector $\boldsymbol{r}$. Note that the surfer often goes back to the query node s with probability c, and thus the proximities are spatially localized around s [26], i.e., scores of nodes tightly connected to s are high, while those of distant nodes are low.

In the following paragraphs, we introduce several existing methods for exact single-source RWR calculation on graphs, with a focus on iterative methods and preprocessing methods. Note that there also exist approximate methods [12, 27, 28, 29], and those for identifying only the top-k nodes with the highest scores [20, 30, 31].

Iterative methods. This approach repeatedly updates RWR scores from the initial ones until convergence. Among various methods, power iteration has been widely utilized due to its simplicity, which is described as follows:

- **Power iteration**: [32] utilized the power iteration method that repeats updating $\boldsymbol{r}$ based on the following equation:

$$\boldsymbol{r}^{(i)} \leftarrow (1-c)\tilde{\mathbf{A}}^\top \boldsymbol{r}^{(i-1)} + c\boldsymbol{q}, \tag{2}$$

  where $\boldsymbol{r}^{(i)}$ denotes $\boldsymbol{r}$ at the i-th iteration. It is repeated until $\boldsymbol{r}$ converges. If $0 < c < 1$, $\boldsymbol{r}$ is guaranteed to converge to a unique solution [33].

Although the iterative approach does not require any computational cost for preprocessing, it exhibits expensive query processing cost (i.e., computational cost per RWR query $\boldsymbol{q}$) due to the repeated matrix-vector calculation for each query.

Preprocessing methods. This approach aims to quickly calculate $\boldsymbol{r}$ for a given query node s based on preprocessed results. From Equation (1), we can represent the problem as solving the following linear system:

$$\left(\mathbf{I}_n - (1-c)\tilde{\mathbf{A}}^\top\right)\boldsymbol{r} = c\boldsymbol{q} \quad \Leftrightarrow \quad \mathbf{H}\boldsymbol{r} = c\boldsymbol{q} \quad \Leftrightarrow \quad \boldsymbol{r} = c\mathbf{H}^{-1}\boldsymbol{q}, \tag{3}$$

where $\mathbf{I}_n$ is an identity matrix of size n, $\mathbf{H} = \mathbf{I}_n - (1-c)\tilde{\mathbf{A}}^\top \in \mathbb{R}^{n\times n}$ is called the random-walk normalized Laplacian matrix with probability $1-c$, and each column of $\mathbf{H}^{-1}$ is the RWR scores w.r.t. each query node.

Note that the inverse of $\mathbf{H}$ always exists since its transpose is a strictly diagonally dominant matrix [34]. However, preprocessing $\mathbf{H}^{-1}$ for large graphs is impractical due to its expensive computational costs (spec., it requires $O(n^3)$ time and $O(n^2)$ space). To overcome the issues, preprocessing approaches focus on precomputing intermediate sub-matrices related to $\mathbf{H}^{-1}$ and computing RWR scores rapidly based on them. These preprocessed matrices are computed only once and can be reused for multiple query nodes while reducing the computational cost per query.

As described later, preprocessing approaches designed for graphs can also be utilized to accelerate RWR computation on hypergraphs within our proposed framework ARCHER. In this paper, we consider the following state-of-the-art methods for computing RWR on graphs, while our framework can be used with any preprocessing-based approaches, such as [15, 21]:

- **BEAR**: [22] developed BEAR, a block elimination approach that efficiently preprocesses sub-matrices related to $\mathbf{H}^{-1}$. For that, they utilized a node reordering technique called SlashBurn[1] [35] using the hub-and-spoke structure to reorder and partition the matrix $\mathbf{H}$. After that, they applied the block elimination [36] to the partitioned sub-matrices for computing $\boldsymbol{r}$.
- **BePI**: [23] proposed BePI, a scalable and memory-efficient method for computing $\boldsymbol{r}$. Although BEAR achieves a fast speed for computing an RWR query, its scalability for larger graphs is limited due to the high cost of the inversion of a sub-matrix inside the block elimination. To resolve the issue, they first utilized SlashBurn[1] to reorder the matrix and then incorporated an iterative approach into the block elimination by replacing the sub-matrix inversion with an iterative linear solver[2].

Note that these two methods have distinct advantages. BePI is more space-efficient and thus can be applied to larger graphs, while BEAR processes each RWR query faster on small datasets. Our experimental results show that the same distinct advantages are observed also in RWR computation on hypergraphs (see Figures 3 and 4).

Other methods to address the substantial cost of the computation of $\mathbf{H}^{-1}$ include employing graph sparsification (i.e., reducing non-zeros of $\mathbf{H}$). For example, [38] developed a spectral sparsification method for directed graphs, demonstrating a strong correlation between the RWR scores computed from the sparsified graph and those obtained from the original graph. Another approach involves approximating $\mathbf{H}$ as an Eulerian Laplacian matrix, followed by the application of a directed Laplacian system-solving algorithm. From the obtained values, approximated values of the original solution can be derived in nearly linear time [39, 40]. These methods differ from BEAR and BePI in that they yield an approximate solution by transforming $\mathbf{H}$ into a computationally efficient form. We plan to explore the incorporation of such approximate computation algorithms into our framework as a part of our future research directions.

Applications. RWR has been extensively utilized in diverse graph mining tasks based on node-to-node similarities on graphs. [14] designed normality scores based on RWR to detect abnormal nodes in a bipartite graph. [15] used RWR to measure the goodness of a match between a query graph and a subgraph. [41] employed RWR for measuring the relevance between a query image and the data images. [13, 42] extended RWR to signed RWR in order to calculate personalized ranking scores in a signed graph. [16] incorporated RWR into graph neural networks (GNNs) to prevent aggregated embeddings from being over-smoothed. The RWR score matrix has been used for augmenting static graphs [17] and dynamic graphs [18] to improve the performance of GNNs.

2.3 Random Walk on Hypergraphs

In this section, we introduce several previous random walk models and other related studies on hypergraphs.

Random walk models on hypergraphs. A typical random walk on a hypergraph [6] repeats (a) selecting an incident hyperedge with probability proportional to edge weights and (b) selecting an incident node uniformly at random. [7] extended the concept of random walk to hypergraphs with edge-dependent node

[1] Let k and n denote the hub selection ratio and the number of nodes, respectively. SlashBurn removes $\lceil kn \rceil$ high-degree nodes (called hubs) from a graph so that it is split into the giant connected component (GCC) and remaining disconnected components (called spokes), and it recursively repeats this process on the GCC. The hubs and spokes are then utilized to construct its reordering permutation (refer to its paper for details). It is used in both BEAR and BePI.

[2] As an iterative solver, BePI employs GMRES [37], a Krylov subspace method, with a preconditioner such as incomplete LU decomposition where the iterative solver converges if its residual is less than error tolerance ϵ.

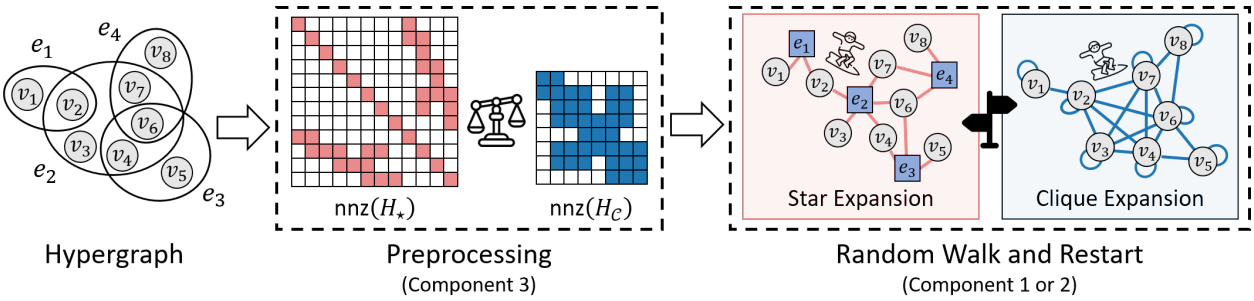


Figure 2: Overview of ARCHER, which consists of 1) a star-expansion-based RWR computation method, 2) a clique-expansion-based RWR computation method, and 3) a preprocessing (including automatic selection) method.

(i.e., vertex) weights (EDNW). Given a hypergraph $G_H = (\mathcal{V}, \mathcal{E}, \omega, \gamma)$ where $\gamma_e(v)$ is the weight of node v depending on edge e, the random walk on G_H is defined as follows:

- **Action 1-1)** For the current node u, the surfer selects a hyperedge e containing node u with probability proportional to $\omega(e)$.
- **Action 1-2)** The surfer moves to node v selected from one of the nodes in the hyperedge e with probability proportional to $\gamma_e(v)$.

In the above model, we set $\gamma_e(u) = 0$ if $u \notin e$. If each node has the same node weight for all of its incident hyperedges (i.e., $\gamma_e(v) = \gamma_{e'}(v)$, $\forall e \neq e' \in \mathcal{E}$), it is called a hypergraph with edge-*independent* node weights (EINW). As described in [7], a random walk on a hypergraph with EINW is equivalent to that on an undirected clique-expanded graph from the hypergraph with some choice of weights; thus, its expressiveness is limited. On the other hand, a random walk on a hypergraph with EDNW is more expressive in that it may not be equivalent to that on any undirected clique-expanded graph.

It should be noticed that even when random walk (with restart) on a hypergraph can be reduced to that on a graph, it may not be computationally optimal to calculate the equivalent random walk (with restart) on a graph. Thus, regardless of this (in)equivalence, it can be useful to develop fast computation methods for random walk (with restart) on hypergraphs.

Applications. [8] employed the random walk to devise a flexible framework for clustering hypergraph data. [9] proposed a local graph cut algorithm using the random walk for product-return prediction on a hypergraph. [10] utilized the random walk for dynamic hypergraph structure learning. [11] proposed HashNWalk which exploits the concept of random walk for detecting anomalous hyperedges in hyperedge streams. Note that these works utilized the concept of random walks on hypergraphs with EDNW, but they did not incorporate the concept of restart. [7] conducted a theoretical analysis of random walks on hypergraphs with EDNW. The authors briefly mentioned that extending this concept to incorporate restart is straightforward, but they did not provide further details. In their work, they utilized RWR for ranking problems on hypergraphs by using naive power iteration, which becomes impractical when dealing with many query nodes on large-scale hypergraphs.

3 Proposed Framework

In this section, we propose ARCHER (Adaptive RWR Computation on Hypergraphs), a novel framework for rapid and space-efficient computation of random walk with restart (RWR) scores on a hypergraph. As depicted in Figure 2, ARCHER consists of three components: (a) star-expansion-based computation methods, (b) clique-expansion-based computation methods, and (c) automatic selection methods. In ARCHER, for a given hypergraph G_H, one between star-expansion-based and clique-expansion-based computation methods is automatically selected based on the number of non-zeros in their resulting matrices (Component 3 in Section 3.4). Then, to leverage preprocessing techniques for fast RWR computation on graphs (e.g., BEAR and BePI), the RWR problem on the hypergraph is converted into that on the star-expanded graph (Component 1 in Section 3.2) or that on the clique-expanded graph (Component 2 in Section 3.3). After

that, the RWR scores with respect to (potentially a large number of) query nodes are computed rapidly by employing a preprocessing-based approach. Note that our framework, ARCHER, can be equipped with any preprocessing-based approaches for RWR computation on graphs.

3.1 Random Walk with Restart on Hypergraphs

First of all, we formally describe the random walk with restart (RWR) model on hypergraphs as follows:

Definition 1 (RWR on a hypergraph)**.** *Given a hypergraph $G_H = (\mathcal{V}, \mathcal{E}, \omega, \gamma)$ and a query node s, a random surfer starts from node s. Then, the surfer takes one of the following actions at each step:*

- ***Action 1) Random walk.*** *The surfer performs the following random walk on G_H with probability $1 - c$.*
  - ***Action 1-1)*** *For the current node u, the random surfer selects a hyperedge e containing node u with probability proportional to $\omega(e)$.*
  - ***Action 1-2)*** *The surfer moves to node v selected from one of the nodes in the hyperedge e with probability proportional to $\gamma_e(v)$.*
- ***Action 2) Restart.*** *The surfer jumps back to the query node s with restart probability c.*

The stationary probability of the surfer visiting a node u is denoted by $\boldsymbol{r}_u$, and the RWR score vector $\boldsymbol{r} \in \mathbb{R}^{n \times 1}$ of all nodes w.r.t. s in G_H is the unique solution of the linear system:

$$\boldsymbol{r} = \underbrace{(1-c)\tilde{\mathbf{R}}^\top \tilde{\mathbf{W}}^\top \boldsymbol{r}}_{\text{Random walk}} + \underbrace{c\boldsymbol{q}}_{\text{Restart}}, \tag{4}$$

where $0 < c < 1$ is the restart probability of a random surfer, $\boldsymbol{q}$ is the RWR query vector, which is the unit vector whose s-th element is 1, and $(\tilde{\mathbf{W}}\tilde{\mathbf{R}})^\top$ is the transition matrix of the random walk on G_H where $\tilde{\mathbf{W}}$ and $\tilde{\mathbf{R}}$ are defined as follows:

- (Regarding **Action 1-1**) $\tilde{\mathbf{W}} = \mathbf{D}_{\mathcal{V}}^{-1}\mathbf{W} \in \mathbb{R}^{n \times m}$ is the row-normalized hyperedge-weight matrix where $\tilde{\mathbf{W}}^\top$ indicates the transition from a node to a hyperedge. $\mathbf{W}$ is the hyperedge-weight matrix, and $\mathbf{D}_{\mathcal{V}} = \texttt{diag}(\mathbf{W}\mathbf{1}_m)$ is the node degree diagonal matrix where $\mathbf{1}_m \in \mathbb{R}^{m \times 1}$ is a column vector of ones.
- (Regarding **Action 1-2**) $\tilde{\mathbf{R}} = \mathbf{D}_{\mathcal{E}}^{-1}\mathbf{R} \in \mathbb{R}^{m \times n}$ is the row-normalized node-weight matrix, where $\tilde{\mathbf{R}}^\top$ indicates the transition from a hyperedge to a node. $\mathbf{R}$ is the node-weight matrix, and $\mathbf{D}_{\mathcal{E}} = \texttt{diag}(\mathbf{R}\mathbf{1}_n)$ is the hyperedge degree diagonal matrix where $\mathbf{1}_n \in \mathbb{R}^n$ is a column vector of ones.

Although the RWR score vector $\boldsymbol{r}$ on G_H can be obtained by repeatedly iterating Equation (4) based on the power iteration method, such an iterative approach is not satisfactory due to its high computational cost per query node, as discussed in Section 2.2. As quickly computing RWR scores for a large number of query nodes is necessary for many applications, in the following sections, we propose two RWR computation methods that provide low cost per query node by preprocessing an input hypergraph, which incurs a one-time cost.

3.2 Component 1. Star-expansion-based Method

We first propose a *star-expansion-based method* that computes the RWR scores on the graph $G_\star$ star-expanded from the hypergraph G_H. For this purpose, we construct a new transition matrix $\tilde{\mathbf{S}}$ as follows:

$$\tilde{\mathbf{S}} = \begin{bmatrix} \mathbf{0} & \tilde{\mathbf{W}} \\ \tilde{\mathbf{R}} & \mathbf{0} \end{bmatrix}, \tag{5}$$

where $\tilde{\mathbf{S}} \in \mathbb{R}^{N \times N}$ is also row-normalized as $\tilde{\mathbf{W}}$ and $\tilde{\mathbf{R}}$ are row-normalized, and $N = n + m$. Note that the sparsity pattern of $\tilde{\mathbf{S}}$ is the same as that of $\mathbf{S}$ which is the star-expanded graph $G_\star$ from G_H as described in Section 2.1 (see the example in Figure 1).

Our star-expansion-based method aims to calculate RWR scores on the new transition matrix $\tilde{\mathbf{S}}$ through the following equation:

$$\boldsymbol{r}_\star = (1 - c_\star)\tilde{\mathbf{S}}^\top \boldsymbol{r}_\star + c_\star \boldsymbol{q}_\star, \tag{6}$$

Algorithm 1 Star-expansion-based Method for RWR on Hypergraphs

function PREPROCESS($\mathbf{R}$, $\mathbf{W}$, c)
  ▷ **Input**: node-weight matrix $\mathbf{R}$, hyperedge-weight matrix $\mathbf{W}$, restart probability c
  ▷ **Output**: set $\mathbf{\Theta}_\star$ of preprocessed matrices
  compute $\mathbf{D}_\mathcal{V}$ and $\mathbf{D}_\mathcal{E}$
  construct $\tilde{\mathbf{S}}$ from $\tilde{\mathbf{W}}$ and $\tilde{\mathbf{R}}$ where $\tilde{\mathbf{R}} = \mathbf{D}_\mathcal{E}^{-1}\mathbf{R}$ and $\tilde{\mathbf{W}} = \mathbf{D}_\mathcal{V}^{-1}\mathbf{W}$
  compute $\mathbf{H}_\star = \mathbf{I}_N - (1 - c_\star)\tilde{\mathbf{S}}^\top$ where $c_\star = 1 - \sqrt{1-c}$ and $N = n + m$
  compute a set $\mathbf{\Theta}_\star$ of preprocessed matrices from $\mathbf{H}_\star$ using a preprocessing method (e.g., BEAR or BePI)
  return $\mathbf{\Theta}_\star$
end function

function QUERY(s, $\mathbf{\Theta}_\star$)
  ▷ **Input**: query node s, set $\mathbf{\Theta}_\star$ of preprocessed matrices
  ▷ **Output**: RWR score vector $\boldsymbol{r}$ w.r.t s in hypergraph G_H
  create $\boldsymbol{q}_\star$ whose s-th entry is 1 and the others are 0
  compute $\boldsymbol{r}_\star$ by querying $\boldsymbol{q}_\star$ based on the preprocessed matrices in $\mathbf{\Theta}_\star$
  decompose $\boldsymbol{r}_\star$ into $\boldsymbol{r}_\mathcal{V}$ and $\boldsymbol{r}_\mathcal{E}$
  compute $\boldsymbol{r} = \frac{c}{c_\star}\boldsymbol{r}_\mathcal{V}$
  return $\boldsymbol{r}$
end function

where $c_\star$ is a modified restart probability from c (i.e., $c_\star = 1 - \sqrt{1-c}$), $\boldsymbol{r}_\star \in \mathbb{R}^{N\times 1}$ is the RWR score vector on $\tilde{\mathbf{S}}$, and $\boldsymbol{q}_\star$ is a modified query vector from $\boldsymbol{q}$ defined as follows:

$$\boldsymbol{r}_\star = \begin{bmatrix} \boldsymbol{r}_\mathcal{V} \\ \boldsymbol{r}_\mathcal{E} \end{bmatrix} \qquad \text{and } \boldsymbol{q}_\star = \begin{bmatrix} \boldsymbol{q} \\ \mathbf{0} \end{bmatrix},$$

where $\boldsymbol{r}_\mathcal{V}$ and $\boldsymbol{r}_\mathcal{E}$ denote the RWR score vectors on nodes and hyperedges, respectively. Once we obtain $\boldsymbol{r}_\mathcal{V}$, the target RWR score vector $\boldsymbol{r}$ is easily converted from $\boldsymbol{r}_\mathcal{V}$ according to the following theorem:

Theorem 1 (Star Expansion Equality)**.** *Suppose $\boldsymbol{r}$ is the RWR score vector on a hypergraph G_H in Equation* (4)*, and $\boldsymbol{r}_\mathcal{V}$ is the sub-vector of $\boldsymbol{r}_\star$, which is the RWR score vector on a star-expanded graph $G_\star$ of $\tilde{\mathbf{S}}$ in Equation* (6)*. Then, the following equality holds:*

$$\boldsymbol{r} = \frac{c}{c_\star}\boldsymbol{r}_\mathcal{V},$$

where $c_\star = 1 - \sqrt{1-c}$ is the modified restart probability, which ranges from 0 to 1 if $0 < c < 1$.

Proof. We rewrite Equation (6) using the definitions of $\tilde{\mathbf{S}}$, $\boldsymbol{r}_\star$ and $\boldsymbol{q}_\star$ as follows:

$$\begin{bmatrix} \boldsymbol{r}_\mathcal{V} \\ \boldsymbol{r}_\mathcal{E} \end{bmatrix} = (1 - c_\star) \begin{bmatrix} \mathbf{0} & \tilde{\mathbf{R}}^\top \\ \tilde{\mathbf{W}}^\top & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_\mathcal{V} \\ \boldsymbol{r}_\mathcal{E} \end{bmatrix} + c_\star \begin{bmatrix} \boldsymbol{q} \\ \mathbf{0} \end{bmatrix}.$$

Then, $\boldsymbol{r}_\mathcal{V}$ and $\boldsymbol{r}_\mathcal{E}$ are represented as:

$$\boldsymbol{r}_\mathcal{V} = (1 - c_\star)\tilde{\mathbf{R}}^\top \boldsymbol{r}_\mathcal{E} + c_\star \boldsymbol{q}, \tag{7}$$

$$\boldsymbol{r}_\mathcal{E} = (1 - c_\star)\tilde{\mathbf{W}}^\top \boldsymbol{r}_\mathcal{V}. \tag{8}$$

By plugging in Equation (8) into Equation (7), we obtain the following equation:

$$\begin{aligned} \boldsymbol{r}_\mathcal{V} &= (1 - c_\star)^2 \tilde{\mathbf{R}}^\top \tilde{\mathbf{W}}^\top \boldsymbol{r}_\mathcal{V} + c_\star \boldsymbol{q} \\ \Leftrightarrow \boldsymbol{r}_\mathcal{V} &= (1 - c_\star)^2 \tilde{\mathbf{P}}^\top \boldsymbol{r}_\mathcal{V} + c_\star \boldsymbol{q}, \end{aligned} \tag{9}$$

where $\tilde{\mathbf{P}} = \tilde{\mathbf{W}}\tilde{\mathbf{R}}$. Note that $c_\star = 1-\sqrt{1-c}$ by its definition, satisfying $(1-c_\star)^2 = (1-c)$. Then, Equation (9) is represented as follows:

$$\begin{aligned} & \boldsymbol{r}_{\mathcal{V}} = (1-c)\tilde{\mathbf{P}}^\top \boldsymbol{r}_{\mathcal{V}} + c_\star \boldsymbol{q} \\ \Leftrightarrow\ & \boldsymbol{r}_{\mathcal{V}} = c_\star \left(\mathbf{I}_n - (1-c)\tilde{\mathbf{P}}^\top\right)^{-1} \boldsymbol{q} \\ \Leftrightarrow\ & \boldsymbol{r}_{\mathcal{V}} = c_\star \mathbf{H}_{\mathcal{C}}^{-1}\boldsymbol{q} = \frac{c_\star}{c}\boldsymbol{r}, \end{aligned}$$

where $\mathbf{H}_{\mathcal{C}} = \mathbf{I}_N - (1-c)\tilde{\mathbf{P}}^\top$, and $\boldsymbol{r} = c\mathbf{H}_{\mathcal{C}}^{-1}\boldsymbol{q}$. This proves the claim $\boldsymbol{r} = \frac{c}{c_\star}\boldsymbol{r}_{\mathcal{V}}$. □

Theorem 1 indicates that the RWR score vector $\boldsymbol{r}$ of Equation (4) can be obtained by solving the RWR problem on the star-expanded graph in Equation (6). Since Equation (6) has the same mathematical form as Equation (1), we can apply preprocessing-based approaches (e.g., BEAR and BePI), which are based on the following linear system:

$$\mathbf{H}_\star \boldsymbol{r}_\star = c_\star \boldsymbol{q}_\star,$$

where $\mathbf{H}_\star = \mathbf{I}_N - (1-c_\star)\tilde{\mathbf{S}}^\top \in \mathbb{R}^{N\times N}$, and $\mathbf{I}_N$ is an identity matrix of size N. Note that $\mathbf{H}_\star$ is invertible, as shown below, and thus the linear system on $\mathbf{H}_\star$ can be solved using preprocessing methods.

Theorem 2 (Invertibility of $\mathbf{H}_\star^\top$)**.** *If $0 < c < 1$, $\mathbf{H}_\star$ is invertible.*

Proof. We first show that $\mathbf{H}_\star^\top$ is strictly diagonally dominant. Note that $\mathbf{H}_\star^\top = \mathbf{I}_N - (1-c_\star)\tilde{\mathbf{S}}$ by its definition, and each entry of $\tilde{\mathbf{S}}$ is non-negative. For each row i, $|\mathbf{H}_{\star_{ii}}^\top| = 1$ because $\tilde{\mathbf{S}}_{ii} = 0$ as shown in Equation (5). For non-diagonal entries of the i-th row of $\mathbf{H}_\star^\top$, $\sum_{j\neq i}|\mathbf{H}_{\star_{ij}}^\top| = 1-c_\star$ since $\tilde{\mathbf{S}}$ is row-normalized. Thus, the following inequality holds for every row i:

$$\sum_{j\neq i}|\mathbf{H}_{\star_{ij}}^\top| = 1 - c_\star < 1 = |\mathbf{H}_{\star_{ii}}^\top|,$$

where $0 < c_\star < 1$ for a given c, indicating $\mathbf{H}_\star^\top$ is strictly diagonally dominant.

The strict diagonal dominance of $\mathbf{H}_\star^\top$ implies its invertibility [34], which, in turn, implies the invertibility of its transposed matrix, $\mathbf{H}_\star$. □

Algorithm 1 summarizes the star-expansion-based method for computing the RWR score vector $\boldsymbol{r}$ w.r.t. a query node s in G_H. The algorithm involves preprocessing and query phases as it adopts a preprocessing-based approach (e.g., BEAR or BePI). Note that the preprocessing phase is run once, while the query phase is run for each query node. In the preprocessing phase, the method first constructs the transition matrix $\tilde{\mathbf{S}}$ (lines 2 and 3). Then, it computes $\mathbf{H}_\star$ (line 4) and preprocesses it by applying a preprocessing-based approach (line 5), resulting in a set $\boldsymbol{\Theta}_\star$ of preprocessed matrices. Whenever a user submits a specific query node s, the query phase computes the RWR score vector $\boldsymbol{r}$ w.r.t. s. Initially, its creates $\boldsymbol{q}_\star$ (line 9), followed by the computation of $\boldsymbol{r}_\star$ (line 12) based on Equation (6). This process employs the query phase of the preprocessing method using the preprocessed results $\boldsymbol{\Theta}_\star$. Based on Theorem 1, the algorithm finally computes the target RWR score vector $\boldsymbol{r}$ (lines 11 and 12).

3.3 Component 2. Clique-expansion-based Method

We propose a *clique-expansion-based method* that computes the RWR scores on the graph $G_{\mathcal{C}}$ clique-expanded from the hypergraph G_H. It explicitly construct the transition matrix $\tilde{\mathbf{P}} = \tilde{\mathbf{W}}\tilde{\mathbf{R}}$ by which Equation (4) becomes

$$\boldsymbol{r} = (1-c)\tilde{\mathbf{P}}^\top \boldsymbol{r} + c\boldsymbol{q}, \tag{10}$$

where the sparsity pattern of $\tilde{\mathbf{P}}$ is the same as that of the adjacency matrix of the clique-expanded graph $G_{\mathcal{C}}$ from G_H as described in Section 2.1 (see the example in Figure 1).

Based on Equation (10), we apply a preprocessing approach to the graph of $\tilde{\mathbf{P}}$ clique-expanded from G_H, which solves the following linear system:

$$\mathbf{H}_{\mathcal{C}}\boldsymbol{r} = c\boldsymbol{q},$$

where $\mathbf{H}_{\mathcal{C}} = \mathbf{I}_n - (1-c)\tilde{\mathbf{P}}^\top \in \mathbb{R}^{n\times n}$ and $\mathbf{I}_n$ is the identity matrix of size n. Note that $\mathbf{H}_{\mathcal{C}}$ is also invertible, which is proven in the following theorem:

Theorem 3 (Invertibility of $\mathbf{H}_{\mathcal{C}}$)**.** *If $0 < c < 1$, $\mathbf{H}_{\mathcal{C}}$ is invertible.*

Proof. We first show that $\mathbf{H}_{\mathcal{C}}^\top$ is strictly diagonally dominant. $\mathbf{H}_{\mathcal{C}}^\top = \mathbf{I}_n - (1-c)\tilde{\mathbf{P}}$ by its definition and each entry of $\tilde{\mathbf{P}}$ is non-negative. For each row i, $|\mathbf{H}_{\mathcal{C}_{ii}}^\top| = 1 - (1-c)\tilde{\mathbf{P}}_{ii}$. Since $\tilde{\mathbf{P}}$ is row-normalized, $\sum_{j\neq i}|\tilde{\mathbf{P}}_{ij}| = 1 - \tilde{\mathbf{P}}_{ii}$. Then, the following inequality holds for every row i:

$$\sum_{j\neq i}|\mathbf{H}_{\mathcal{C}_{ij}}^\top| = (1-c)(1-\tilde{\mathbf{P}}_{ii}) = (1-(1-c)\tilde{\mathbf{P}}_{ii}) - c < 1 - (1-c)\tilde{\mathbf{P}}_{ii} = |\mathbf{H}_{\mathcal{C}_{ii}}^\top|,$$

where $0 < c < 1$. This indicates that $\mathbf{H}_{\mathcal{C}}^\top$ is strictly diagonally dominant.

The strict diagonal dominance of $\mathbf{H}_{\mathcal{C}}^\top$ implies its invertibility [34], which, in turn, implies the invertibility of its transposed matrix, $\mathbf{H}_{\mathcal{C}}$. □

The clique-expansion-based method is summarized in Algorithm 2, which consists of preprocessing and query phases. In the preprocessing phase, it first explicitly builds the transition matrix $\tilde{\mathbf{P}} \in \mathbb{R}^{n\times n}$ (lines 2 and 3). Then, the algorithm computes the matrix $\mathbf{H}_{\mathcal{C}}$ (line 4). By applying a preprocessing method, it processes $\mathbf{H}_{\mathcal{C}}$ and obtains the preprocessed results $\boldsymbol{\Theta}_{\mathcal{C}}$ (line 5). In the query phase, it creates the RWR query vector $\boldsymbol{q}$ (line 9) and then computes the RWR score vector $\boldsymbol{r}$ by querying $\boldsymbol{q}$ using the preprocessed results $\boldsymbol{\Theta}_{\mathcal{C}}$ (line 10). The query phase is initiated whenever a user submits a query node.

The time and space complexities of both clique- and star-expansion-based methods can be directly derived from the complexities of RWR computation methods (e.g., BEAR [22] and BePI [23]) and the definitions of clique- and star-expansions. When employing BePI as the RWR computation method, although the complexities involve many terms related to graph structures, empirically, processing time, space cost, and query time are largely influenced by the number of edges after expansion, specifically, $\texttt{nnz}(\mathbf{H}_{\mathcal{C}})$ and $\texttt{nnz}(\mathbf{H}_{\star})$ in clique- and star-expansion-based computations, respectively. For detailed empirical results, refer to Appendix E. This empirical tendency is utilized in the subsequent subsection for the automatic selection between clique- and star-expansion-based methods.

3.4 Component 3. Automatic Selection Method

As described in Sections 3.2 and 3.3, our clique- and star-expansion-based methods allow for fast RWR computation on the hypergraph G_H by leveraging a preprocessing approach. Interestingly, the preprocessed matrices $\mathbf{H}_{\star}$ and $\mathbf{H}_{\mathcal{C}}$ have very different characteristics. For example, $\mathbf{H}_{\mathcal{C}}$ may have a large number of non-zeros because each hyperedge e is replaced with a clique of all nodes in e, which can exert a bad effect on scalability as preprocessed results are densified. On the other hand, $\mathbf{H}_{\star} \in \mathbb{R}^{N\times N}$ can be relatively sparse, but it is of large dimension. Recall that $N = n + m$ where n and m are the numbers of nodes and edges, respectively. As a result, the relative time and space required to preprocess $\mathbf{H}_{\star}$ and $\mathbf{H}_{\mathcal{C}}$ heavily depends on datasets. For example, if $n \ll m$, preprocessing a small matrix such as $\mathbf{H}_{\mathcal{C}} \in \mathbb{R}^{n\times n}$ can be computationally advantageous even though it is dense.

Thus, we further develop an *automatic selection method* for choosing one between clique- and star-expansion-based methods so that the chosen method brings out the best performance of a preprocessing method. Various data statistics of a hypergraph can be considered to design a criterion based on which one method is chosen. Among them, our strategy is to utilize the number of non-zeros of a matrix to be preprocessed under a hypothesis that the performance of a preprocessing method is likely to be affected by non-zero entries as it exploits the sparsity of the preprocessed matrix for efficiency. We empirically prove the effectiveness of our strategy compared to various statistics in Section 5.5.

Based on the criterion, our framework selects the star-expansion-based method if the following predicate satisfies:

$$\texttt{nnz}(\mathbf{H}_{\mathcal{C}}) > \texttt{nnz}(\mathbf{H}_{\star}) \tag{11}$$

Algorithm 2 Clique-expansion-based Method for RWR on Hypergraphs

function PREPROCESS($\mathbf{R}$, $\mathbf{W}$, c)
  ▷ **Input**: node-weight matrix $\mathbf{R}$, hyperedge-weight matrix $\mathbf{W}$, restart probability c
  ▷ **Output**: set $\mathbf{\Theta}_{\mathcal{C}}$ of preprocessed matrices
  compute $\mathbf{D}_{\mathcal{V}}$ and $\mathbf{D}_{\mathcal{E}}$
  compute $\tilde{\mathbf{P}} = \tilde{\mathbf{W}}\tilde{\mathbf{R}}$ where $\tilde{\mathbf{W}} = \mathbf{D}_{\mathcal{V}}^{-1}\mathbf{W}$ and $\tilde{\mathbf{R}} = \mathbf{D}_{\mathcal{E}}^{-1}\mathbf{R}$
  compute $\mathbf{H}_{\mathcal{C}} = \mathbf{I}_n - (1-c)\tilde{\mathbf{P}}^{\top}$
  compute a set $\mathbf{\Theta}_{\mathcal{C}}$ of preprocessed matrices from $\mathbf{H}_{\mathcal{C}}$ using a preprocessing method (e.g., BEAR or BePI)
  return $\mathbf{\Theta}_{\mathcal{C}}$
end function

function QUERY(s, $\mathbf{\Theta}_{\mathcal{C}}$)
  ▷ **Input**: query node s, set $\mathbf{\Theta}_{\mathcal{C}}$ of preprocessed matrices
  ▷ **Output**: RWR score vector $\boldsymbol{r}$ w.r.t s in hypergraph G_H
  create $\boldsymbol{q}$ whose s-th entry is 1 and the others are 0
  compute $\boldsymbol{r}$ by querying $\boldsymbol{q}$ based on the preprocessed matrices in $\mathbf{\Theta}_{\mathcal{C}}$
  return $\boldsymbol{r}$
end function

Algorithm 3 ARCHER: Adaptive RWR Computation on Hypergraphs

function PREPROCESS($\mathbf{R}$, $\mathbf{W}$, c)
  ▷ **Input**: node-weight matrix $\mathbf{R}$, hyperedge-weight matrix $\mathbf{W}$, restart probability c
  ▷ **Output**: set $\mathbf{\Theta}$ of preprocessed matrices
  set $\texttt{nnz}(\mathbf{H}_{\mathcal{C}})$ to $\texttt{nnz}(\mathbf{WR})$ by Equation (12)
  set $\texttt{nnz}(\mathbf{H}_{\star})$ to $n + m + 2\sum_{v\in\mathcal{V}} \bar{d}(v)$ by Equation (13)
  if $\texttt{nnz}(\mathbf{H}_{\mathcal{C}}) > \texttt{nnz}(\mathbf{H}_{\star})$ **then**
    $\mathbf{\Theta} \leftarrow$ PREPROCESS($\mathbf{R}$, $\mathbf{W}$, c) of Algorithm 1 ▷ star expansion
  else
    $\mathbf{\Theta} \leftarrow$ PREPROCESS($\mathbf{R}$, $\mathbf{W}$, c) of Algorithm 2 ▷ clique expansion
  end if
  return $\mathbf{\Theta}$
end function

function QUERY(s, $\mathbf{\Theta}$)
  ▷ **Input**: query node s, set $\mathbf{\Theta}$ of preprocessed matrices
  ▷ **Output**: RWR score vector $\boldsymbol{r}$ w.r.t s in hypergraph G_H
  if $\mathbf{\Theta}$ is from the star-expansion-based method **then**
    $\boldsymbol{r} \leftarrow$ QUERY(s, $\mathbf{\Theta}$) of Algorithm 1 ▷ star expansion
  else
    $\boldsymbol{r} \leftarrow$ QUERY(s, $\mathbf{\Theta}$) of Algorithm 2 ▷ clique expansion
  end if
  return $\boldsymbol{r}$
end function

where $\texttt{nnz}(\cdot)$ returns the number of non-zeros of an input matrix. Otherwise, our method selects the clique-expansion-based method. Note that counting $\texttt{nnz}(\mathbf{H}_{\mathcal{C}})$ and $\texttt{nnz}(\mathbf{H}_{\star})$ takes a very short time compared to the total running time mostly consumed by non-trivial operations, such as reordering and matrix multiplications of the preprocessing methods [22, 23]. While the empirical computation time is already very small, further optimization allows us to calculate Equation (11) in $O(\sum_{e\in\mathcal{E}}|e|^2)$ time with $O(n)$ extra space, as explained in Appendix D.

3.5 Ultimate Framework: ARCHER

By putting all of the components together, we develop ARCHER, our ultimate framework for fast computation of RWR on hypergraphs, and the procedure in it is summarized in Algorithm 3. In the preprocessing phase, ARCHER first computes $\texttt{nnz}(\mathbf{H}_{\mathcal{C}})$ and $\texttt{nnz}(\mathbf{H}\star)$ (lines 2 and 3), which are utilized to select either clique- or star-expansion-based methods. Based on the criterion in Equation (11), ARCHER chooses one of the methods and executes the corresponding preprocessing step (lines 4-8) to obtain a set $\mathbf{\Theta}$ of preprocessed matrices. In the query phase, for each query node s, ARCHER performs the query step depending on the selected method (lines 12-16), using the preprocessed matrices $\mathbf{\Theta}$, to compute the RWR score vector $\boldsymbol{r}$.

It is important to note that our framework ARCHER can be equipped with any pre-processing-based RWR computation method (e.g., BEAR and BePI), which is used inside Algorithms 1 and 2, and the total time and space complexity of ARCHER depends on the chosen method. Different methods offer distinct advantages, as demonstrated empirically in Section 5.

4 Application to Anomaly Detection

In this section, we present an application of RWR scores on hypergraphs for the purpose of anomaly detection. The empirical effectiveness of this approach is demonstrated in Section 5.6.

Given a hypergraph, this application aims to detect anomalous hyperedges that deviate from ordinary group interactions. Inspired from [14], which uses RWR for anomaly detection on graphs, we measure a normality score of a hyperedge based on relevance scores provided by hypergraph RWR. Our intuition is that if a hyperedge e is normal, the relevance scores between any pair of nodes in e should be high. Specifically, we define the normality score $ns(e)$ as follows:

$$ns(e) = \frac{1}{|e|(|e|-1)} \sum_{u \in e} \sum_{v \in e \setminus \{u\}} \boldsymbol{r}_{u \to v}$$

where $\boldsymbol{r}_{u \to v}$ is the RWR score of the node v w.r.t. the query node u. In other words, it is the average pairwise relevance scores between nodes in the hyperedge e. If the normality score is low, then the hyperedge is considered anomalous. Note that we need a fast computation method such as ARCHER for this task because it requires RWR scores for many query nodes.

More Applications. We have conducted a study on another application focused on the task of *node retrieval*, which is described in Appendix B.

5 Experiments

In this section, we evaluate the performance of ARCHER and compare it with other baselines for computing RWR on hypergraphs. We aim to answer the following questions from the experiments:

- **Q1. Preprocessing Time (Section 5.2).** How long do ARCHER and the two computation methods composing it take for preprocessing?
- **Q2. Space Cost (Section 5.3).** How much memory space do they require for their preprocessed results?
- **Q3. Query Time (Section 5.4).** How quickly do they process RWR queries?
- **Q4. Automatic Selection Method (Section 5.5).** How precisely does our automatic selection strategy decide an appropriate computation method for a given hypergraph?
- **Q5. Application to Anomaly Detection (Section 5.6).** Can we achieve more accurate anomaly detection on hypergraphs using RWR scores, compared to existing approaches?

5.1 Experimental Settings

We describe our experimental settings, including machines, methods, datasets, and parameters.

Machines. All experiments are conducted on a workstation with AMD Ryzen 9 3900X and 128GB memory.

Table 2: Data statistics of real-world hypergraphs. n and m are the numbers of nodes and hyperedges, respectively. The average and maximum sizes of hyperedges are denoted by $\text{avg}_{e\in\mathcal{E}}|e|$ and $\max_{e\in\mathcal{E}}|e|$, respectively. The density is defined by m/n, and the overlapness [43] is defined by $\Sigma_{e\in\mathcal{E}}|e|/n$.

Dataset	n	m	$\text{avg}_{e\in\mathcal{E}}\|e\|$	$\max_{e\in\mathcal{E}}\|e\|$	Density	Overlapness
EEN	143	10,883	2.47	37	76.12	188.21
EEU	998	234,760	2.39	40	234.09	559.80
SB	294	29,157	7.96	99	99.17	789.62
HB	1,494	60,987	20.47	399	40.82	835.79
WAL	88,860	69,906	6.59	25	0.79	5.81
TRI	172,738	233,202	3.12	85	1.35	4.21
AM	55,700	105,655	8.12	555	1.90	15.41
YP	25,252	25,656	18.2	649	1.02	18.50
TW	81,305	70,097	25.2	1205	0.86	21.75
COH	1,014,734	1,812,511	1.32	925	1.75	2.32
COG	1,256,385	1,590,335	2.80	284	1.26	3.53
COD	1,924,991	3,700,067	2.79	280	1.92	5.35
THU	125,602	192,947	1.80	14	1.54	2.76
THM	176,445	719,792	2.24	21	4.08	9.13
THS	2,675,955	11,305,343	2.23	67	4.22	9.56
ML1	3,533	6,038	95.3	1435	1.71	162.83
ML10	10,472	69,816	84.3	3375	6.67	562.02
ML20	22,884	138,362	88.1	4168	6.05	532.93

Methods. For experiments evaluating the computational performance, we compare ARCHER with using always one expansion method (star- or clique-expansion-based method). ARCHER and the baselines are equipped with BEAR [22] or BePI [23], state-of-the-art preprocessing methods for RWR on graphs. We also compare ARCHER with the power iteration methods on star- and clique-expanded graphs in Equations (6) and (10), respectively. We use our MATLAB implementation of the power iterations, and we use the source code of the authors for BEAR[3] and BePI[4], which are also implemented in MATLAB.

For anomaly detection, we consider the following two hypergraph-based anomaly-detection methods as baseline approaches:

- **LSH-A [44]:** For each incoming hyperedge, it computes the approximate frequency, which represents the number of previous hyperedges that are similar to the new one, in the aspect of common nodes shared. This is used to measure the unexpectedness of the hyperedge; intuitively, the hyperedge can be considered unexpected if it is novel and significantly different from previous hyperedges (i.e., low frequency). The approximation scheme is efficiently implemented in hyperedge streams by using Locality Sensitive Hashing (LSH) [45]. Specifically, it computes the MinHash signature with k_h hash functions and performs LSH with b bands. Then it scores the hyperedge with the similarity between the signature of previous hyperedges. LSH-A takes $O(k_h|e| + b + b\lceil\frac{mb}{B}\rceil + 1)$ time per hyperedge, and we use our Python implementation of LSH-A.
- **HashNWalk [11]:** In HashNWalk, each hyperedge is hashed into M buckets called supernodes using each of k_h hash functions. Based on the transition probability (i.e., random walk of length 1), HashNWalk calculates the proximity between supernodes. Whenever a new hyperedge emerges, HashNWalk updates the proximity between the supernodes in it and compares it with the previous proximity to compute the anomaly score of the hyperedge. In scoring, HashNWalk incorporates the hyperparameter α to control the degree of emphasis placed on recent hyperedges. HashNWalk takes $O(k_h|e| + k_h \min(M, |e|)^2)$ time per hyperedge, and we use the official implementation of HashNWalk in C++.[5]

[3] `https://datalab.snu.ac.kr/bear`
[4] `https://datalab.snu.ac.kr/bepi`
[5] `https://github.com/geon0325/HashNWalk`

Datasets. We conduct extensive experiments on eighteen real-world hypergraphs [1, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55], whose statistics are summarized in Table 2. We provide details of each dataset in Appendix A. The source code and datasets used in this paper are available at `https://github.com/jaewan01/ARCHER`.

Parameters. For the experiments of preprocessing and query costs, we set the restart probability c to 0.05, which has been widely used in previous work [12, 22, 23]. For BEAR, we set the hub selection ratio k of the reordering method to 0.001 as in [22]. For BePI, we set the hub selection ratio k of the reordering method to 0.2, which is used for large graphs in [23] (see Footnote 1 for the usage of k). The error tolerance ϵ for the power iteration and BePI is set to 10^{-9}. We set the time and memory limits for preprocessing to 12 hours and 128 GB, respectively.

We also perform careful hyperparameter tuning for the aforementioned anomaly detection methods. For LSH-A, we measure performance with different numbers of bands in LSH signatures ($b \in \{2, 4, 8\}$) and the length of LSH signatures ($l \in \{2, 4, 8\}$) and report the best result obtained. For HashNWalk, we adopt a similar setting as described in [11]. Specifically, we set the hyperparameter α in the kernel function to 0.98. Additionally, we conduct a search for the optimal values of the number of hash functions k_h and the number of buckets M in the following ranges: (a) $k_h \in \{10, 15, 20\}$ and $M \in \{10, 20, 30\}$ for the email-Enron dataset (b) $k_h \in \{8, 10, 12\}$ and $M \in \{60, 80, 100\}$ for the senate-bills dataset, and (c) $k_h \in \{8, 10, 12\}$ and $M \in \{100, 150, 200\}$ for the house-bills dataset. We report the best result achieved for each dataset.

5.2 Preprocessing Time

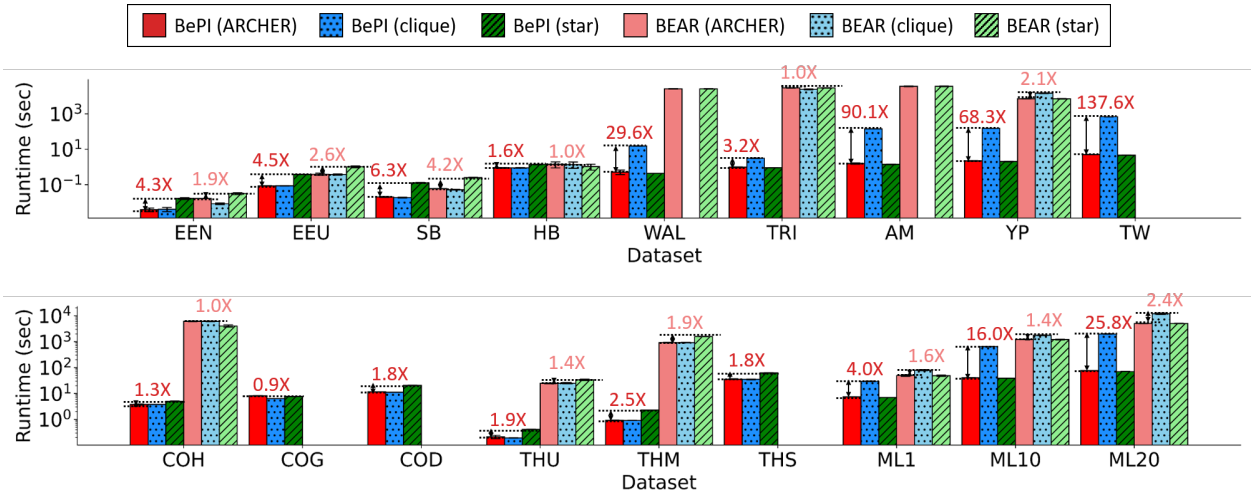
We evaluate the performance of ARCHER in terms of preprocessing time. BEAR and BePI are used as preprocessing techniques. For each method, we report the average preprocessing time of 10 experiments. Note that the iterative methods are excluded in this experiment because they do not require preprocessing. Figure 3a shows the preprocessing time of all tested methods on 18 real-world hypergraphs.

We first compare ARCHER with BEAR and each expansion-based method with BEAR. ARCHER with BEAR preprocesses hypergraphs up to 4.2× faster than applying BEAR to the star-expanded graph (see the results on the SB dataset) and up to 2.4× faster than applying BEAR to the clique-expanded graph (see the results on the ML20 dataset). Note that the clique-expansion-based method with BEAR cannot preprocess medium-sized datasets such as WAL and AM because clique expansion produces a too-dense matrix that BEAR cannot handle. For larger datasets such as TW, COG, COD, and THS, all methods with BEAR fail due to their limited scalability. On the other hand, ARCHER with BePI provides better scalability for preprocessing, and it successfully preprocesses all the datasets, showing up to 137.6× faster than applying BePI to the clique-expanded graph (see the results on the TW dataset) and up to 4.5× faster than applying BePI to the star-expanded graph (see the results on the EEU dataset).

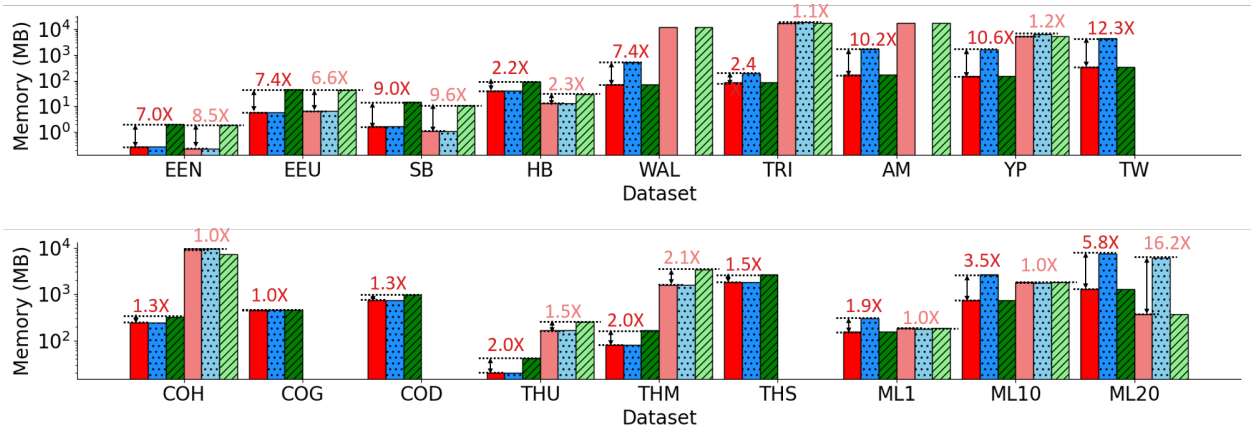
These results imply the complementary nature of the clique-expansion-based methods and the star-expansion-based methods. Specifically, they exhibit relative advantages on different hypergraphs, and these advantages are significant. This underscores the importance of making careful choices between the two methods, as ARCHER does. Our results regarding space cost and query time in the following subsections further reinforce the importance of this selection, although we do not repeat the same discussion within those subsections.

5.3 Space Cost

We analyze the space cost of ARCHER with each preprocessing technique compared to that of other methods. The iterative methods are excluded from the comparison because they do not produce preprocessed results beyond the size of the original data. We measure the memory size for storing preprocessed results in MB. Figure 3b shows the space cost of each method. ARCHER with BEAR uses up to 16.2× less memory than applying BEAR to the clique-expanded graph (see the results on the ML20 dataset) and up to 9.6× less memory than applying BEAR to the star-expanded graph (see the results on the SB dataset). Furthermore, the space cost of ARCHER with BePI is up to 12.3× less than applying BePI to the clique-expanded graph (see the results on the TW dataset) and up to 9.0× less than applying BePI to the clique-expanded graph (see the results on the SB dataset).



(a) Preprocessing time



(b) Space usage for preprocessed results

Figure 3: Preprocessing costs for various calculation methods of RWR on hypergraphs in terms of (a) preprocessing time and (b) space usage. For preprocessing times, the error bars indicate ±1 standard deviation. However, in many datasets, the error bars are so small that they are practically invisible. As shown in the figures, using ARCHER requires up to 137.6× less preprocessing time and 16.2× less space than using always one expansion method. Results are omitted if the corresponding methods ran out of time (> 12 hours) or out of memory (> 128 GB) during preprocessing.

5.4 Query Time

We examine the computational efficiency of ARCHER in processing RWR queries, compared to other baselines, including power-iteration methods. For every method, we measure the average query processing time for the same 30 query nodes. Figure 4 shows the results on 18 real-world hypergraphs. ARCHER with BEAR answers the RWR queries up to 16.3× faster than applying BEAR to the star-expanded graph (see the results on the EEU dataset) and up to 1.3× faster than applying BEAR to the clique-expanded graph (see the results on the ML20 dataset) For ARCHER with BePI, its query time is up to 218.8× less than applying BePI to the star-expanded graph (see the results on the EEU dataset) and up to 4.0× faster than applying BePI to the clique-expanded graph (see the results on the AM dataset). Note that regardless of the preprocessing techniques used, ARCHER significantly outperforms both power-iteration methods.

The experimental results regarding preprocessing and query costs imply that we should consider different aspects of the preprocessing methods when choosing one of them. If query cost matters more than prepro-

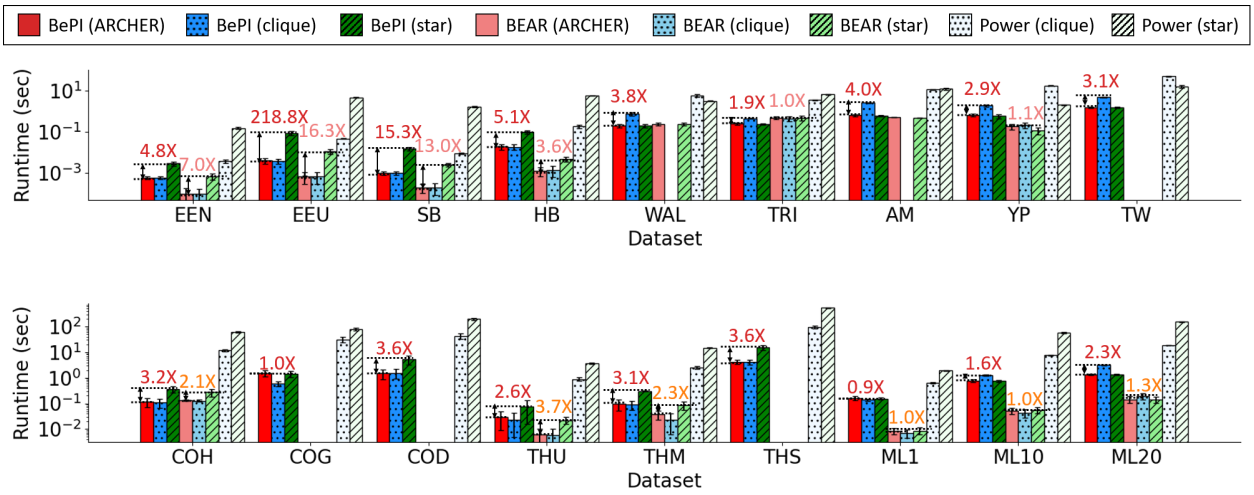


Figure 4: Query time for various calculation methods of RWR on hypergraphs. The error bars indicate ±1 standard deviation. Using ARCHER takes up to 218.8× less query time than using always one expansion method. Results are omitted if the corresponding methods ran out of time (> 12 hours) or out of memory (> 128 GB) during preprocessing.

cessing cost, BEAR is a good choice because its query speed is faster than that of BePI, especially on small datasets, as shown in Figure 4. On the other hand, BePI is required for scalable RWR computation on larger hypergraphs because BEAR fails to process such hypergraphs, as discussed in Section 5.2.

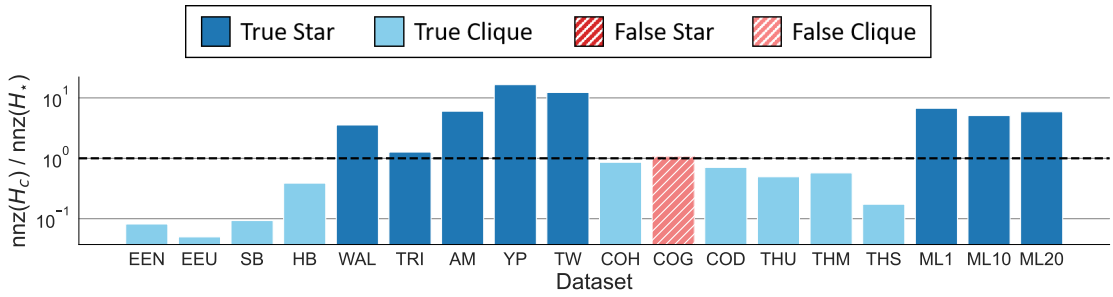
5.5 Automatic Selection Method

We investigate the effectiveness of the proposed criterion in our automatic selection method in ARCHER, compared to other potential criteria based on major statistics of hypergraphs, including average hyperedge size, density, and overlapness [43], which are summarized in Table 2.

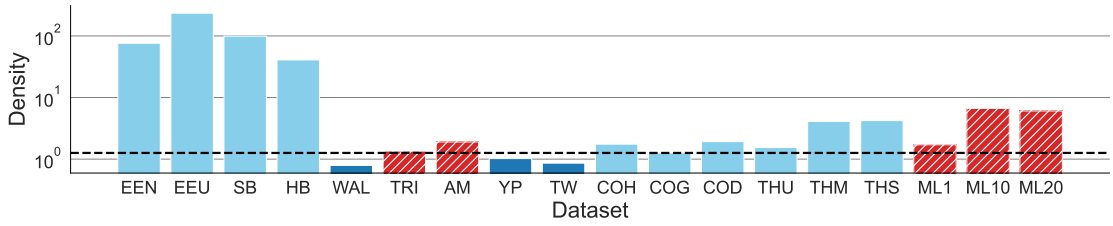
To evaluate the effectiveness of each criterion, we set the ground-truth label between star- and clique-expansion methods for each dataset depending on which expansion method leads to a shorter preprocessing time of BePI. The reason for labeling the datasets in this manner is that only BePI successfully preprocesses all the datasets, and the preprocessing time of each method is distinctly different, as shown in Figure 3a. Our method selects the star method if $\texttt{nnz}(\mathbf{H}_{\mathcal{C}})/\texttt{nnz}(\mathbf{H}_{\star}) > 1$; otherwise, it picks the clique method, i.e., the threshold for our suggested method is 1 in Figure 5a. For the other criteria, we selected the threshold values that maximize the number of correct selections. Specifically, we set the threshold for density in Figure 5b to 1.26, the threshold for overlapness in Figure 5c to 15.41, and the threshold for the average size of hyperedges in Figure 5d to 3.12.

Figure 5 demonstrates that, when using our proposed criterion, we can choose the better expansion method for 17 (out of 18) datasets, while the number of wrong choices increases when we use the other criteria. This result empirically confirms our hypothesis that the performances of preprocessing methods heavily depend on the number of non-zeros in the preprocessed matrix.

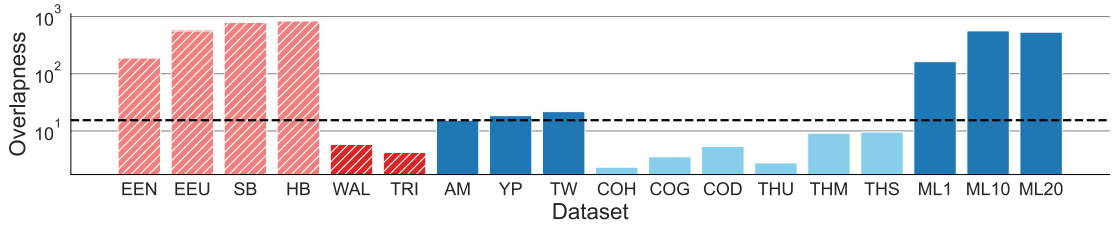
We further analyze the correlation between two ratios: (1) the ratio of $\texttt{nnz}(\mathbf{H}_{\mathcal{C}})$ and $\texttt{nnz}(\mathbf{H}_{\star})$, and (2) the ratio of (preprocessing, space, and query) costs of the clique- and star-expansion-based methods, across various datasets when BePI is used. In Figure 6, we observe a strong positive correlation between the non-zero-count ratio and the cost ratio for each aspect of computation. This indicates that the cost in each aspect is closely related to the number of non-zero entries in the matrix being processed. Additionally, the positioning of data points in each plot demonstrates the effectiveness of our suggested method selection. Data points located in the lower left area indicate the correct selection of the clique method, while those in the upper right area indicate the correct selection of the star method. This demonstrates that our suggested selection approach is effective for most of the tested datasets.



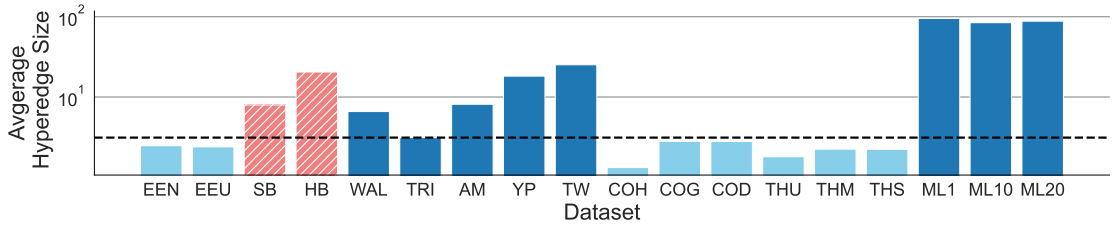
(a) Proposed relative ratio $\texttt{nnz}(\mathbf{H}_C)/\texttt{nnz}(\mathbf{H}_\star)$ (17/18 correct)



(b) Density (13/18 correct)



(c) Overlapness (12/18 correct)



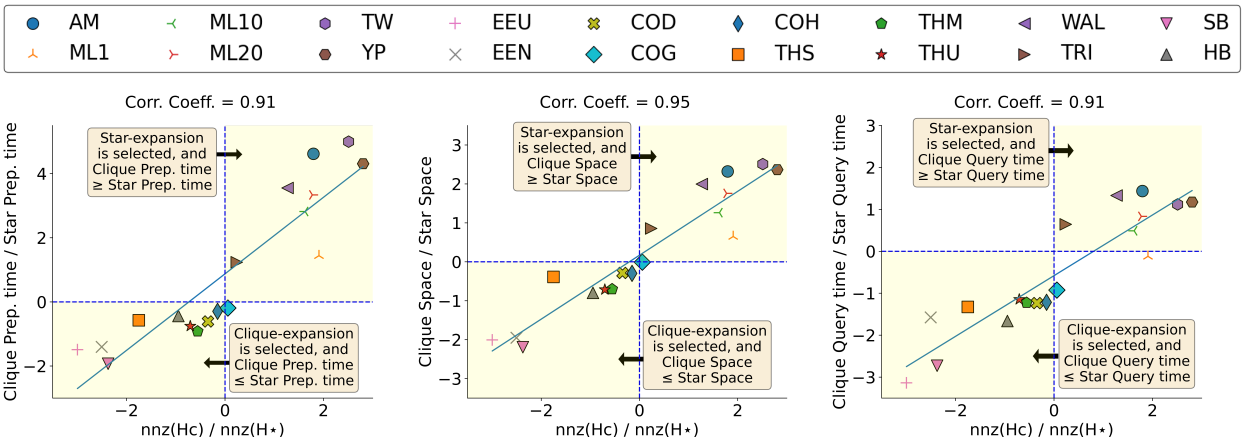
(d) Average hyperedge size (16/18 correct)

Figure 5: Effectiveness of various data statistics in selection between the star- and clique-expansion-based methods. BePI is used as the preprocessing approach. Each dashed line indicates the best threshold for the corresponding criterion. Note that, when using our proposed criterion, we can choose the better expansion method for 17 (out of 18) datasets, while the number of wrong choices increases when we use the other criteria.

5.6 Application to Anomaly Detection

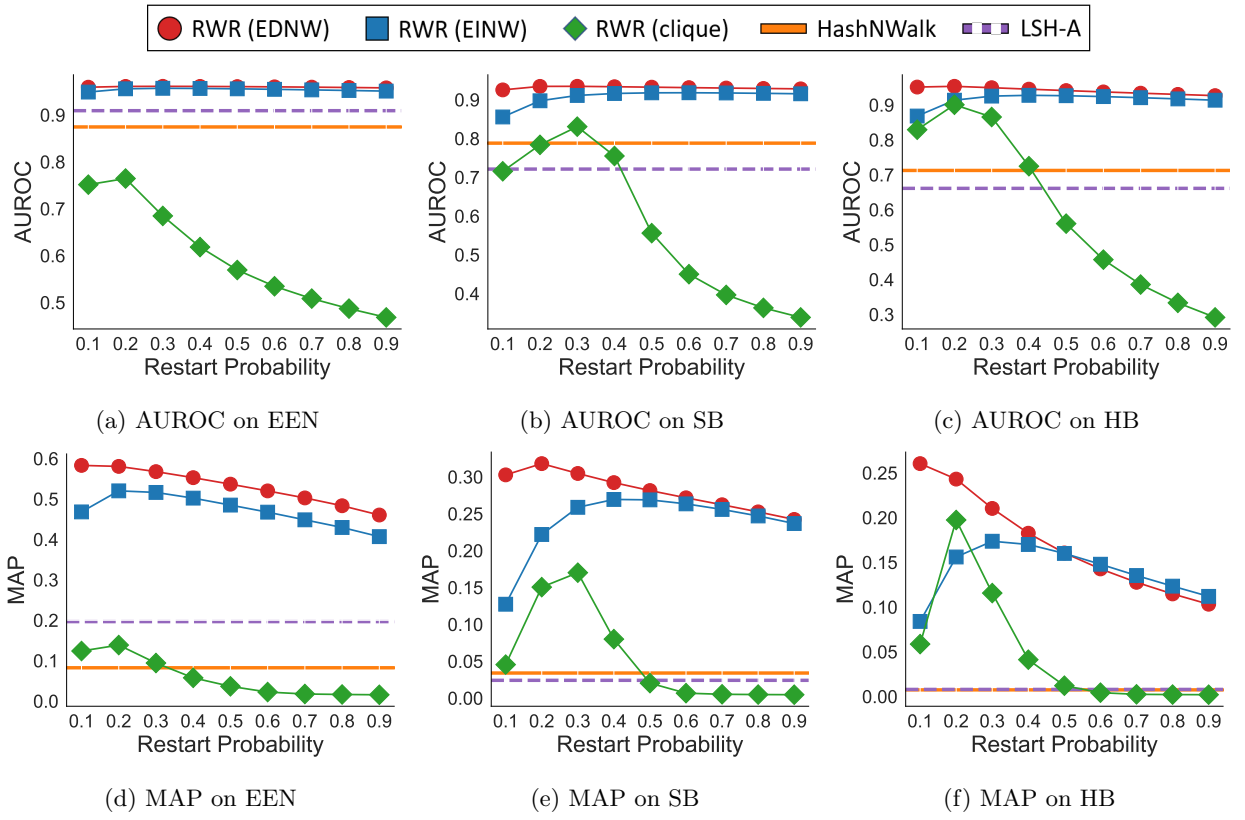
In this section, we evaluate the effectiveness of RWR scores in detecting anomalies on hypergraphs. Refer to Section 4 for a detailed procedure on how we utilize RWR scores for anomaly detection.

Settings. We conduct this experiment on the EEN (email-Enron), SB (senate-bills), and HB (house-bills) datasets, which are small enough for all compared methods to terminate. As the original datasets do not contain anomalous hyperedges, we synthetically generate them by injecting *unexpected hyperedges* following [11]. Specifically, we randomly select a hyperedge $e \in E$, and then create a hyperedge by replacing half



(a) Preprocessing Time (b) Space Cost (c) Query Time

Figure 6: Correlation between two ratios on a natural logarithmic scale: (1) the ratio of $\texttt{nnz}(\mathbf{H}_{\mathcal{C}})$ and $\texttt{nnz}(\mathbf{H}_{\star})$, and (2) the ratio of the costs of the clique- and star-expansion-based methods in terms of (a) preprocessing time, (b) space cost, and (c) query time. BePI is used as the preprocessing technique. We report the Pearson correlation coefficient for each scatter plot.



(a) AUROC on EEN (b) AUROC on SB (c) AUROC on HB

(d) MAP on EEN (e) MAP on SB (f) MAP on HB

Figure 7: Anomaly detection performance on hypergraphs in terms of AUROC and MAP. RWRs using weights, especially EDNW, are more accurate than the other methods, including RWR on unweighted clique-expanded graphs.

nodes in e with random nodes. We repeat this process until t hyperedges are generated.

We consider three models of RWR on hypergraphs: 1) RWR using EDNW, 2) RWR using EINW, and 3)

naive RWR. Note that the datasets used in the experiment do not contain explicit weights. For EDNW, we make the weights to be edge-dependent by setting $\gamma_e(v) = \bar{d}(v)^{-\beta}$, where $\bar{d}(v)$ denotes the unweighted degree of node v. That is, we adopt the principle that, since high-degree nodes are present in multiple hyperedges, they are likely to have a reduced impact within each hyperedge. We then set $\beta = 0.5$ and $\omega(e) = 1$ (refer to Appendix C for the selection of $\beta = 0.5$). In the case of EINW (edge-independent node weights), we set $\gamma_e(v) = 1$ and $\omega(e) = 1$. To check the effect of those weights, we further compare naive RWR that computes RWR scores on the unweighted clique-expanded graph from the input hypergraph. We vary the restart probability c from 0.1 to 0.9 by 0.1.

We compare those RWR models with LSH-A and HashNWalk, anomaly detection methods on hypergraphs. While they are originally designed for hypergraphs with timestamps, the used datasets are static, and thus we assign a random timestamp to each hyperedge (spec., we randomly order the hyperedges and use their orders as timestamps) when testing them.

Results. Figure 7 demonstrates the performance of each model for detecting the anomalous (unexpected) hyperedges in terms of AUROC and MAP. As shown in the figure, RWR using EDNW performs best, implying that it is beneficial to utilize the edge-dependent node weights. Even RWR using EINW outperforms HashNWalk and LSH-A. Naive RWR performs worst as the weights of nodes and hyperedges are all disregarded.

6 Conclusion and Future Directions

In this work, we consider random walk with restart (RWR) on hypergraphs after formally defining it (Definition 1). Then, we propose ARCHER (Algorithm 3) for its rapid and space-efficient computation. ARCHER is composed of two RWR computation methods (Algorithms 1 and 2) that are based on clique- and star-expanded graphs, respectively, of the input hypergraph. Since their relative performance heavily depends on datasets, ARCHER is equipped with a lightweight automatic method for selecting one between them. Using 18 real-world hypergraphs, we substantiate the speed and space efficiency of ARCHER (Figures 3 and 4), revealing that these qualities are attributed to the complementary nature of the two RWR computation methods and the accuracy of the automatic selection method (Figure 5). In addition, we introduce anomaly detection as an application of RWR on hypergraphs and show the empirical effectiveness of RWR on it (Figure 7).

As potential directions for future work, we intend to extend our framework to incorporate approximate RWR computation algorithms, as discussed in Section 2.2. Furthermore, the present automatic selection algorithm in Section 3.4 is grounded in empirical observations, in which we plan to develop theoretically grounded yet efficient selection algorithms.

Declarations

Funding

This work was supported by National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2020R1C1C1008296) (No. NRF-2021R1C1C1008526) and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-00075, Artificial Intelligence Graduate School Program (KAIST)) (No. 2021-0-02068, Artificial Intelligence Innovation Hub).

Competing interests

The authors have no relevant financial or non-financial interests to disclose.

References

[1] Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon Kleinberg. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences*, 115(48), nov 2018. doi: 10.1073/pnas.1800683115.

[2] Manh Tuan Do, Se-eun Yoon, Bryan Hooi, and Kijung Shin. Structural patterns and generative models of real-world hypergraphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 176–186. ACM, aug 2020. doi: 10.1145/3394486.3403060.

[3] Geon Lee, Jaemin Yoo, and Kijung Shin. Mining of real-world hypergraphs: Patterns, tools, and generators. In *Proceedings of the 29th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 5811–5812. ACM, 2023. doi: 10.1145/3580305.3599567.

[4] Cazamere Comrie and Jon Kleinberg. Hypergraph ego-networks and their temporal evolution. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 91–100, dec 2021. doi: 10.1109/icdm51629.2021.00019.

[5] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, apr 1998. doi: 10.1016/s0169-7552(98)00110-x.

[6] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS)*, page 1601–1608, 2006. doi: 10.7551/mitpress/7503.003.0205.

[7] Uthsav Chitra and Benjamin Raphael. Random walks on hypergraphs with edge-dependent vertex weights. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 1172–1181, 09–15 Jun 2019. doi: 10.48550/arXiv.1905.08287.

[8] Koby Hayashi, Sinan G. Aksoy, Cheong Hee Park, and Haesun Park. Hypergraph random walks, laplacians, and clustering. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM)*, pages 495–504, oct 2020. doi: 10.1145/3340531.3412034.

[9] Jianbo Li, Jingrui He, and Yada Zhu. E-tail product return prediction via hypergraph-based local graph cut. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, page 519—527, jul 2018. doi: 10.1145/3219819.3219829.

[10] Zizhao Zhang, Haojie Lin, and Yue Gao. Dynamic hypergraph structure learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3162–3169, jul 2018. doi: 10.24963/ijcai.2018/439.

[11] Geon Lee, Minyoung Choe, and Kijung Shin. HashNWalk: Hash and random walk based anomaly detection in hyperedge streams. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2129–2137, jul 2022. doi: 10.24963/ijcai.2022/296.

[12] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Random walk with restart: fast solutions and applications. *Knowledge and Information Systems*, 14(3):327–346, jul 2007. doi: 10.1007/s10115-007-0094-2.

[13] Jinhong Jung, Woojeong Jin, Lee Sael, and U Kang. Personalized ranking in signed networks using signed random walk with restart. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 973–978, dec 2016. doi: 10.1109/icdm.2016.0122.

[14] Jimeng Sun, Huiming Qu, Deepayan Chakrabarti, and Christos Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM)*, pages 418–425, 2005. doi: 10.1109/ICDM.2005.103.

[15] Hanghang Tong, Christos Faloutsos, Brian Gallagher, and Tina Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 737–746, aug 2007. doi: 10.1145/1281192.1281271.

[16] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations (ICLR)*, 2019. doi: 10.48550/arXiv.1810.05997.

[17] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *Advances in neural information processing systems (NeurIPS)*, 2019. doi: 10.48550/arXiv.1911.05485.

[18] Jong-whi Lee and Jinhong Jung. Time-aware random walk diffusion to improve dynamic graph learning. In *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, 2023. doi: 10.1609/aaai.v37i7.26021.

[19] Runhui Wang, Sibo Wang, and Xiaofang Zhou. Parallelizing approximate single-source personalized PageRank queries on shared memory. *The VLDB Journal*, 28(6):923–940, oct 2019. doi: 10.1007/s00778-019-00576-7.

[20] Guanhao Hou, Xingguang Chen, Sibo Wang, and Zhewei Wei. Massively parallel algorithms for personalized pagerank. *Proceedings of the VLDB Endowment*, 14(9):1668–1680, may 2021. doi: 10.14778/3461535.3461554.

[21] Yasuhiro Fujiwara, Makoto Nakatsuji, Makoto Onizuka, and Masaru Kitsuregawa. Fast and exact top-k search for random walk with restart. *Proceedings of the VLDB Endowment*, 5(5):442–453, jan 2012. doi: 10.14778/2140436.2140441.

[22] Kijung Shin, Jinhong Jung, Sael Lee, and U. Kang. Bear: Block elimination approach for random walk with restart on large graphs. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1571–1585, may 2015. doi: 10.1145/2723372.2723716.

[23] Jinhong Jung, Namyong Park, Sael Lee, and U Kang. BePI. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD)*, pages 789–804, may 2017. doi: 10.1145/3035918.3035950.

[24] Liang Sun, Shuiwang Ji, and Jieping Ye. Hypergraph spectral learning for multi-label classification. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 668–676, aug 2008. doi: 10.1145/1401890.1401971.

[25] J.Y. Zien, M.D.F. Schlag, and P.K. Chan. Multi-level spectral hypergraph partitioning with arbitrary vertex sizes. *ITCSDI*, 18(9):1389–1399, 1999. doi: 10.1109/iccad.1996.569592.

[26] Huda Nassar, Kyle Kloster, and David F. Gleich. Strong localization in personalized PageRank vectors. In *Algorithms and Models for the Web Graph (WAW)*, pages 190–202, 2015. doi: 10.1007/978-3-319-26784-5_15.

[27] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. Unifying the global and local approaches: An efficient power iteration with forward push. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*, pages 1996–2008, jun 2021. doi: 10.1145/3448016.3457298.

[28] Dandan Lin, Raymond Chi-Wing Wong, Min Xie, and Victor Junqiu Wei. Index-free approach with theoretical guarantee for efficient random walk with restart query. In *IEEE 36th International Conference on Data Engineering (ICDE)*, pages 913–924, apr 2020. doi: 10.1109/icde48307.2020.00084.

[29] Sibo Wang, Renchi Yang, Runhui Wang, Xiaokui Xiao, Zhewei Wei, Wenqing Lin, Yin Yang, and Nan Tang. Efficient algorithms for approximate single-source personalized PageRank queries. *ACM Transactions on Database Systems*, 44(4):1–37, oct 2019. doi: 10.1145/3360902.

[30] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. Topppr: Top-k personalized pagerank queries with precision guarantees on large graphs. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*, pages 441–456, may 2018. doi: 10.1145/3183713.3196920.

[31] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. Fora: Simple and effective approximate single-source personalized pagerank. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 505–514, aug 2017. doi: 10.1145/3097983.3098072.

[32] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[33] Amy N. Langville and Carl D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, Princeton, 2006. ISBN 9781400830329. doi: doi: 10.1515/9781400830329.

[34] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012. doi: 10.1017/CBO9780511810817.

[35] U. Kang and Christos Faloutsos. Beyond 'caveman communities': Hubs and spokes for graph compression and mining. In *2011 IEEE 11th International Conference on Data Mining (ICDM)*, pages 300–309, 2011. doi: 10.1109/ICDM.2011.26.

[36] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004. doi: 10.1017/CBO9780511804441.

[37] Lloyd N Trefethen and David Bau. *Numerical linear algebra*, volume 181. Siam, 2022. doi: 10.1137/1.9780898719574.

[38] Ying Zhang, Zhiqiang Zhao, and Zhuo Feng. A unified approach to scalable spectral sparsification of directed graphs. *arXiv:1812.04165*, 2018. doi: 10.48550/arXiv.1812.04165.

[39] Michael B. Cohen, Jonathan Kelner, Rasmus Kyng, John Peebles, Richard Peng, Anup B. Rao, and Aaron Sidford. Solving directed laplacian systems in nearly-linear time through sparse lu factorizations. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 898–909, 2018. doi: 10.1109/FOCS.2018.00089.

[40] Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Aaron Sidford, and Adrian Vladu. Faster algorithms for computing the stationary distribution, simulating random walks, and more. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 583–592, 2016. doi: 10.1109/FOCS.2016.69.

[41] Songhao Zhu, Liming Zou, and Baojie Fang. Content based image retrieval via a transductive model. *Journal of Intelligent Information Systems*, 42(1):95–109, jul 2013. doi: 10.1007/s10844-013-0257-4.

[42] Jinhong Jung, Woojeong Jin, and U Kang. Random walk-based ranking in signed social networks: model and algorithms. *Knowledge and Information Systems*, 62(2):571–610, may 2019. doi: 10.1007/s10115-019-01364-z.

[43] Geon Lee, Minyoung Choe, and Kijung Shin. How do hyperedges overlap in real-world hypergraphs? - patterns, measures, and generators. In *Proceedings of the Web Conference 2021 (WWW)*, pages 3396–3407, apr 2021. doi: 10.1145/3442381.3450010.

[44] Stephen Ranshous, Mandar Chaudhary, and Nagiza F. Samatova. Efficient outlier detection in hyperedge streams using MinHash and locality-sensitive hashing. In *Complex Networks & Their Applications VI*, pages 105–116, nov 2017. doi: 10.1007/978-3-319-72150-7_9.

[45] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011. doi: 10.1017/CBO9781139924801.

[46] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June (Paul) Hsu, and Kuansan Wang. An overview of microsoft academic service (MAS) and applications. In *Proceedings of the 24th International Conference on World Wide Web (WWW)*, pages 519–527, may 2015. doi: 10.1145/2740908.2742839.

[47] Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 555–564, aug 2017. doi: 10.1145/3097983.3098069.

[48] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution. *ACM Transactions on Knowledge Discovery from Data*, 1(1):2–es, mar 2007. doi: 10.1145/1217299.1217301.

[49] Ilya Amburg, Nate Veldt, and Austin Benson. Clustering in graphs and hypergraphs with categorical edge labels. In *Proceedings of The Web Conference 2020 (WWW)*, pages 706–717, apr 2020. doi: 10.1145/3366423.3380152.

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

[51] James H. Fowler. Connecting the congress: A study of cosponsorship networks. *Political Analysis*, 14 (4):456–487, 2006. doi: 10.1093/pan/mpl002.

[52] James H. Fowler. Legislative cosponsorship networks in the US house and senate. *Social Networks*, 28 (4):454–465, oct 2006. doi: 10.1016/j.socnet.2005.11.003.

[53] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, 2019. doi: 10.18653/v1/d19-1018.

[54] Julian McAuley and Jure Leskovec. Discovering social circles in ego networks. *arXiv:1210.8182*, 2013. doi: 10.48550/arXiv.1210.8182.

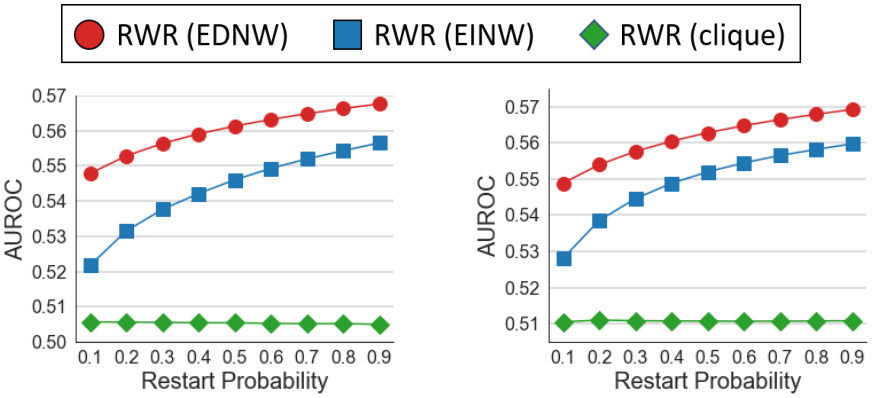
[55] F. Maxwell Harper and Joseph A. Konstan. The MovieLens datasets. *ACM Transactions on Interactive Intelligent Systems*, 5(4):1–19, dec 2015. doi: 10.1145/2827872.

A Experimental Datasets

We provide a brief description of the datasets used in this paper.

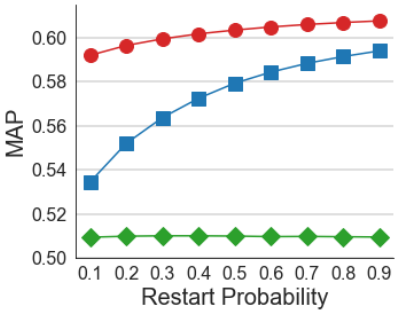
- **EEN and EEU.**[6] These hypergraphs represent sets of email addresses on emails in Enron (EEN) and a European research institution (EEU) where users are nodes and the group of the sender and all receivers of each email is a hyperedge.
- **HB and SB.**[6] These represent co-sponsorships of bills in the House of Representatives (HB) and the Senate (SB) where US Congresspersons are nodes, and groups of sponsors and co-sponsors of bills are hyperedges.
- **WAL.**[6] This is a hypergraph where products are nodes and hyperedges are sets of co-purchased products at Walmart.
- **TRI.**[6] This is a hypergraph where nodes are accommodations (mostly hotels) and hyperedges are sets of accommodations that a user performed "click-out" during the same browsing session at Trivago.
- **COD, COG, and COH.**[6] These are co-authorship hypergraphs where authors are nodes and each hyperedge represents the authors of a publication recorded on DBLP (COD), Geology (COG), and History (COH).
- **THS, THM, and THU).**[6] These are hypergraphs where users are nodes and each hyperedge represents the group of users associated with a thread at StackOverflow (THS), MathStackOverflow (THM), and AskUbuntu (THU).

[6] `https://www.cs.cornell.edu/~arb/data/`

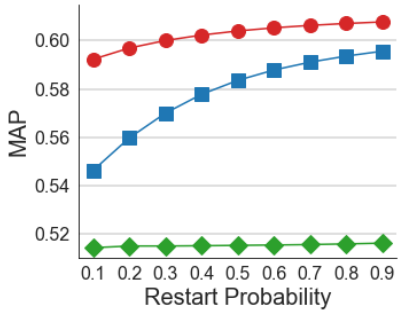


(a) AUROC on SB

(b) AUROC on HB



(c) MAP on SB



(d) MAP on HB

Figure 8: Similar-node-retrieval performance on hypergraphs in terms of AUROC and MAP. The hypergraph RWR using EDNW provides the best accuracy among all the baselines in the SB and HB datasets.

- **AM.**[7] This is a hypergraph of Amazon (AM) product reviews (spec., those categorized as Movies & TV) where users are nodes and a group of products reviewed by the same user is a hyperedge. Each user has at least 5 reviews.
- **YP.**[8] This is a hypergraph of user ratings on locations (e.g., hotels and restaurants) at Yelp (YP) where users are nodes and a group of locations a user rated is a hyperedge. Ratings higher than 3 are considered.
- **TW.**[9] This is a hypergraph of social relationships on Twitter (TW) where users are nodes and each hyperedge represents a group of users that compose a 'circle' (or 'list') together on Twitter.
- **ML1, ML10, and ML20.**[10] These hypergraphs represent interactions of movies at MovieLens with different sizes of 1M (ML1), 10M (ML10), and 20M (ML20) movie ratings where nodes are movies and a group of movies a user rated is a hyperedge. Ratings higher than 3 are considered.

B Application to Node Retrieval

In this section, we introduce an application of RWR scores on hypergraphs for the task of node retrieval. Additionally, we evaluate the empirical effectiveness of this approach.

Similar Node Retrieval. Given a hypergraph and a query node s, this task is to search for nodes structurally similar to the query node. Specifically, we measure node-to-node proximities for s and use them as ranking scores to sort all nodes except s in the order of the scores. If nodes with the same class of the

[7] https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2
[8] https://www.yelp.com/dataset
[9] https://snap.stanford.edu/data/ego-Twitter.html
[10] https://grouplens.org/datasets/movielens/

query node are ranked high, structurally similar nodes are successfully retrieved, which can be evaluated by ranking metrics such as AUROC and MAP. For this task, we compute the hypergraph RWR scores $\boldsymbol{r}$ w.r.t. query node s through ARCHER, and utilize the scores for ranking.

Settings. We conduct this experiment on the SB (senate-bills) and HB (house-bills) datasets, which contain binary node labels. The RWR models used in Section 5.6 are also used for this task. To introduce edge-dependent node weights (EDNW), we set $\gamma_e(v) = \bar{d}(v)^{-\beta}$, where $\bar{d}(v)$ represents the unweighted degree of node v. We then set $\beta = 1.0$ and $\omega(e) = 1$ (refer to Appendix C for the selection of $\beta = 1.0$). For EINW, we set $\gamma_e(v) = 1$ and $\omega(e) = 1$. We vary the restart probability c from 0.1 to 0.9 by 0.1.

Results. Figure 8 shows the experimental results on the node retrieval task in terms of AUROC and MAP. As shown in the figure, the RWR using EDNW shows the best performance, especially with high values of restart probability c, among all tested methods. Note that the RWR using EDNW outperforms that using EINW and naive RWR, indicating the edge-dependent node weights are useful also for this task.

C Experiments on Edge-dependent Node Weights for Applications

In this section, we provide the experimental results regarding the effectiveness of the edge-dependent node weights for applications.

Anomaly Detection. For anomaly detection in Section 5.6, we set edge-dependent node weights $\gamma_e(v) = \bar{d}(v)^{-\beta}$ for hypergraph RWR. For the experiment, we assess the performance of RWR by varying two parameters: β and the restart probability c. Specifically, we explore different values of β within the range of $\{0.5, 1.0, 2.0\}$, and we also vary c between 0.1 and 0.9 in increments of 0.1. Figure 9 shows the results, $\beta = 0.5$ generally yields the best performance across the tested datasets.

Similar Node Retrieval. For node retrieval in Appendix B, we also set $\gamma_e(v) = \bar{d}(v)^{-\beta}$ for hypergraph RWR. We test the node-retrieval performance of RWR by varying the values of β and c. Specifically, the list of values tested for β is $\{0.5, 1.0, 2.0\}$, while the range for c spans from 0.1 to 0.9. Figure 10 shows the results, and $\beta = 1.0$ leads to the best performance in most cases.

D Counting of the Number of Non-zero Entries

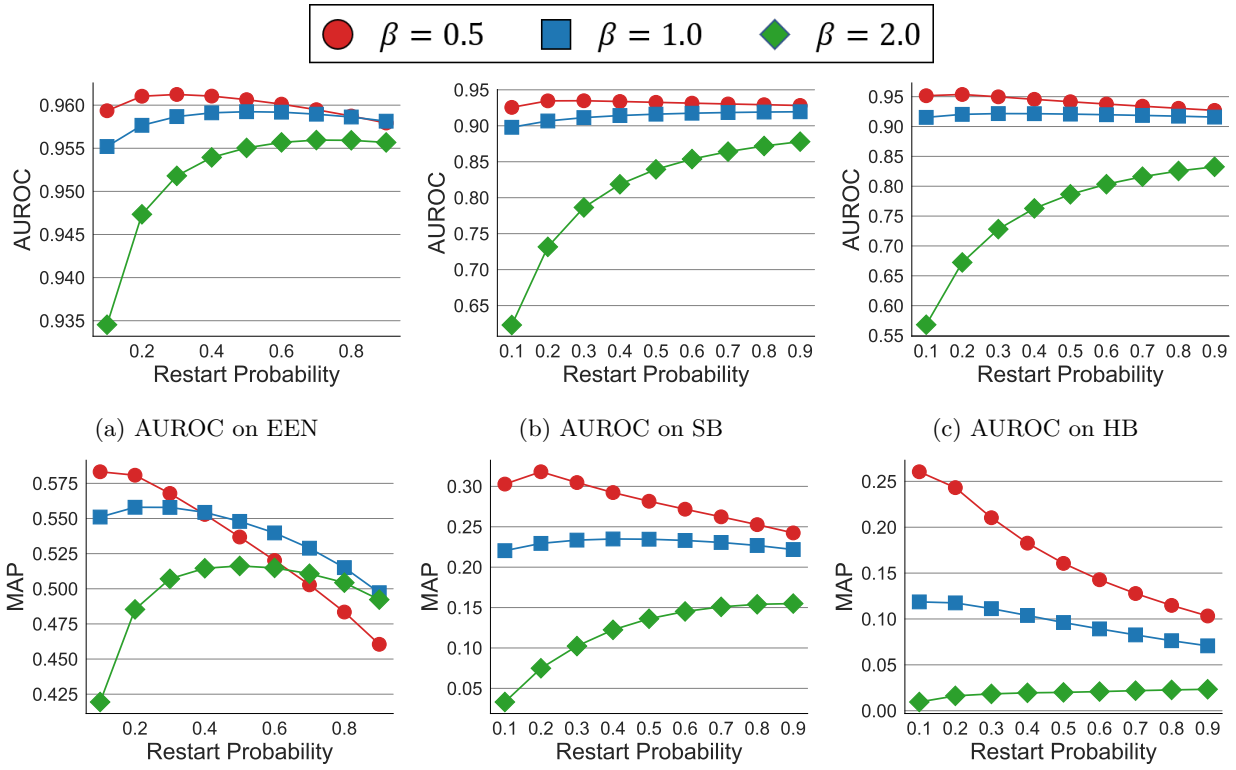
In this section, we discuss how to compute $\texttt{nnz}(\mathbf{H}_{\mathcal{C}})$ and $\texttt{nnz}(\mathbf{H}_{\star})$ rapidly and space-efficiently. They are used in Equation (11) by ARCHER to select one between clique- and star-expansion-based methods.

Calculation of $\texttt{nnz}(\mathbf{H}_{\mathcal{C}})$. While it is possible to naively count the number of non-zeros in $\mathbf{H}_{\mathcal{C}} = \mathbf{I}_n - (1-c)\tilde{\mathbf{P}}^\top$, materializing $\mathbf{H}_{\mathcal{C}}$ typically requires more space than the input data due to its relatively high density. Hence, we suggest a more efficient way based on the following property regarding $\mathbf{H}_{\mathcal{C}}$:

$$\texttt{nnz}(\mathbf{H}_{\mathcal{C}}) = \texttt{nnz}(\tilde{\mathbf{P}}) \tag{12}$$

where $\tilde{\mathbf{P}} = \tilde{\mathbf{W}}\tilde{\mathbf{R}}$. The equality is from the fact that the diagonal entries of $\tilde{\mathbf{P}}$ are non-zeros because $\tilde{\mathbf{P}}$ involves the transition probability that moves from each node v to one of its hyperedges, and goes back to v. Note the sparsity pattern of $\tilde{\mathbf{P}}$ is the same as that of the adjacency matrix of the clique-expanded graph $G_{\mathcal{C}}$ (with additional self-loops on every node) of the hypergraph G_H. Thus, we can calculate $\texttt{nnz}(\tilde{\mathbf{P}})$ without materializing $\tilde{\mathbf{P}}$, by directly counting the edges that are clique-expanded from each hyperedge.

Algorithm 4 summarizes the procedure for computing $\texttt{nnz}(\mathbf{H}_{\mathcal{C}})$. For each node v (line 2), we find every node u that appears together with v in at least one hyperedge (line 5). Whenever we find such u, it is equivalent to finding an edge (v, u), and thus we increment the count accordingly (line 7). Note that we maintain a set C of such nodes to prevent duplicated counting (lines 3 and 8). Regardless of the input, Algorithm 4 requires $O(|C|) = O(n)$ extra space to maintain the set C. The time complexity is $O(\sum_{v\in\mathcal{V}}\sum_{e\in E(v)}|e|) = O(\sum_{e\in\mathcal{E}}|e|^2)$ because it requires $|e|$ operations for each node in e.



(a) AUROC on EEN (b) AUROC on SB (c) AUROC on HB

(d) MAP on EEN (e) MAP on SB (f) MAP on HB

Figure 9: Anomaly detection performance on hypergraphs in terms of AUROC and MAP with different edge-dependent node weights, i.e., $\gamma_e(v) = \bar{d}(v)^{-\beta}$. When $\beta = 0.5$, it provides the best accuracy.

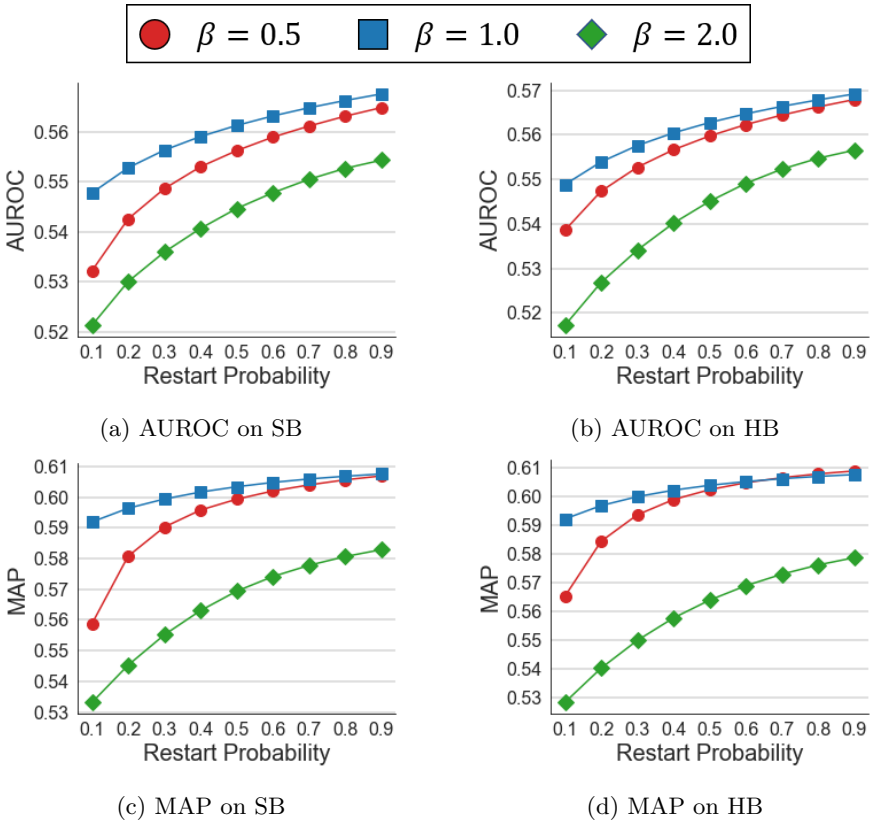
Algorithm 4 Counting the number of non-zeros of $\mathbf{H}_{\mathcal{C}}$

Input: sets $\mathcal{V}$ and $\mathcal{E}$ of nodes and hyperedges in G_H, resp.
Output: number of non-zeros of $\mathbf{H}_{\mathcal{C}}$

set $\texttt{nnz} \leftarrow 0$
for $v \in \mathcal{V}$ **do**
  $C \leftarrow \emptyset$ ▷ C is a set of nodes that appear together in one or more hyperedges
  for $e \in E(v)$ **do** ▷ $E(v)$ is the set of hyperedges incident to node v
    for $u \in e$ **do**
      if $u \notin C$ **then**
        $\texttt{nnz} \leftarrow \texttt{nnz} + 1$ ▷ (v, u) is counted
        $C \leftarrow C \cup \{u\}$
      end if
    end for
  end for
end for
return nnz

Calculation of $\texttt{nnz}(\mathbf{H}_\star)$. Similarly, $\texttt{nnz}(\mathbf{H}_\star)$ can also be efficiently calculated based on the following equalities:

$$\begin{aligned}\texttt{nnz}(\mathbf{H}_\star) &= n + m + \texttt{nnz}(\tilde{\mathbf{S}}) \\ &= n + m + \texttt{nnz}(\tilde{\mathbf{W}}) + \texttt{nnz}(\tilde{\mathbf{R}}) \\ &= n + m + \texttt{nnz}(\mathbf{W}) + \texttt{nnz}(\mathbf{R}),\end{aligned} \tag{13}$$



(a) AUROC on SB

(b) AUROC on HB

(c) MAP on SB

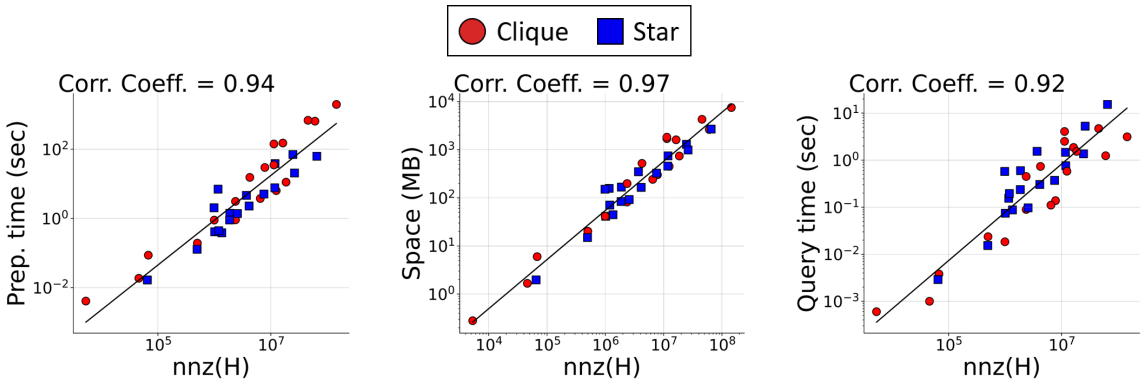
(d) MAP on HB

Figure 10: Similar-node-retrieval performance in terms of AUROC and MAP with different edge-dependent node weights, i.e., $\gamma_e(v) = \bar{d}(v)^{-\beta}$. When $\beta = 1.0$, it provides better performance in most of the cases.

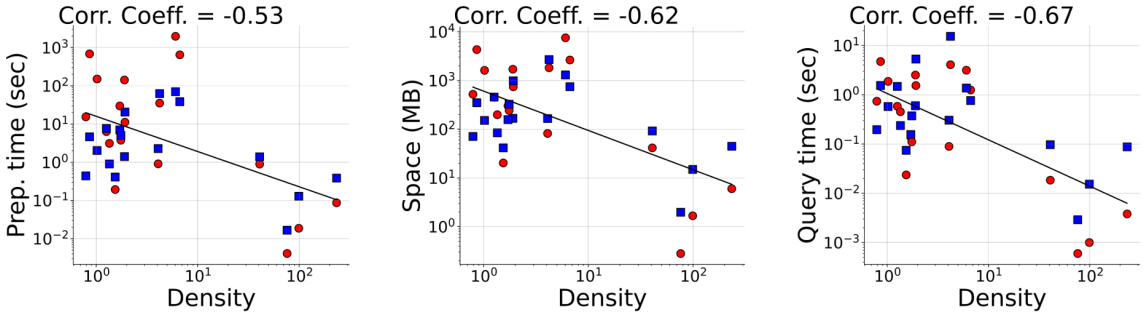
where $\mathbf{H}_\star = \mathbf{I}_N - (1-c)\tilde{\mathbf{S}}^\top$. Note that $\mathbf{I}_N$ is the identity matrix of size $N = n + m$, occupying $n + m$ non-zeros in $\mathbf{H}_\star$. The matrix $\tilde{\mathbf{S}}$ consists of $\tilde{\mathbf{W}}$ and $\tilde{\mathbf{R}}$ as shown in Equation (5), and their sparsity patterns are the same as $\mathbf{W}$ and $\mathbf{R}$. The time complexity of this approach is dominated by that of counting the numbers of non-zero entries in $\mathbf{W}$ and $\mathbf{R}$. If $\mathbf{W}$ and $\mathbf{R}$ are in a sparse matrix format, the number of their non-zero entries can be computed in $O(\texttt{nnz}(\mathbf{W}) + \texttt{nnz}(\mathbf{R})) = O(\sum_{v\in\mathcal{V}} \bar{d}(v)) = O(\sum_{e\in\mathcal{E}} |e|)$ time and even in $O(1)$ time in some formats (e.g., compressed sparse row). With the exception of the inputs (i.e., $\mathbf{W}$ and $\mathbf{R}$), this approach requires a constant amount of additional space.

E Correlation between Data Statistics and Costs of BePI

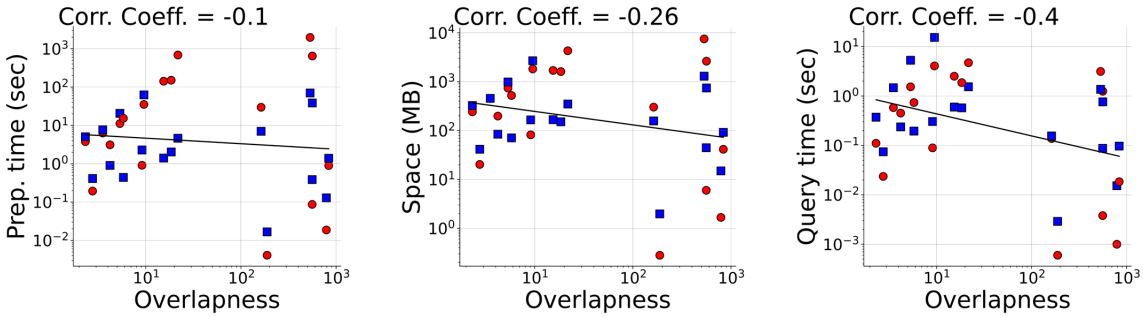
In this section, we empirically investigate the correlations between basic data statistics and the costs of BePI, which ARCHER employs for RWR computation. As the data statistics, we use $\texttt{nnz}(\mathbf{H})$ (i.e., $\texttt{nnz}(\mathbf{H}_\mathcal{C})$ and $\texttt{nnz}(\mathbf{H}_\star)$ in clique- and star-expansion-based computations, respectively), density, overlapness, and average hyperedge size. As the costs of the clique- and star-expansion-based computation of BePI, we consider preprocessing time, space cost, and query time. The results obtained across all the datasets (refer to Appendix A) for both clique- and star-expansion-based computations are presented in Figure 11. As shown in Figure 11a, there exists a strong positive correlation between $\texttt{nnz}(\mathbf{H})$ and the costs for the calculation of RWR. For other statistics (see Figures 11b, 11c, and 11d), there is no noticeable correlation between the statistics and the costs.



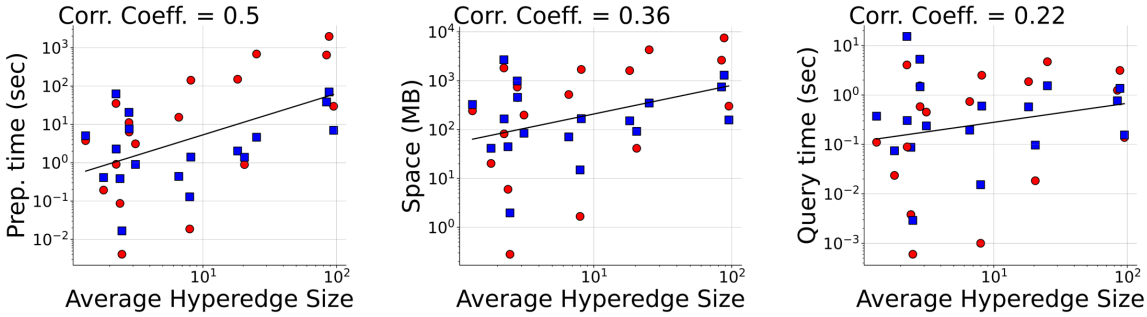
(a) Correlation between $\texttt{nnz}(\mathbf{H})$ (i.e., $\texttt{nnz}(\mathbf{H}_{\mathcal{C}})$ and $\texttt{nnz}(\mathbf{H}_{\star})$ in clique- and star-expansion-based computations, respectively) and RWR computation costs



(b) Correlation between density and RWR computation costs



(c) Correlation between overlapness and RWR computation costs



(d) Correlation between average hyperedge size and RWR computation costs

Figure 11: Correlations between (a) basic data statistics and (b) the costs of RWR computation on hypergraphs in terms of preprocessing time, space cost, and query time. BePI is used for RWR computation. We report the Pearson correlation coefficient for each scatter plot. Note that there is a strong positive correlation between $\texttt{nnz}(\mathbf{H})$ and the costs, whereas other statistics do not exhibit such a correlation.