HashNWalk: Hash and Random Walk Based Anomaly Detection in Hyperedge Streams

Geon Lee , **Minyoung Choe** and **Kijung Shin**
Kim Jaechul Graduate School of AI, KAIST, Seoul, South Korea
{geonlee0325, minyoung.choe, kijungs}@kaist.ac.kr

Abstract

Sequences of group interactions, such as emails, online discussions, and co-authorships, are ubiquitous; and they are naturally represented as a stream of hyperedges. Despite their broad potential applications, anomaly detection in hypergraphs (i.e., sets of hyperedges) has received surprisingly little attention, compared to that in graphs. While it is tempting to reduce hypergraphs to graphs and apply existing graph-based methods, according to our experiments, taking higher-order structures of hypergraphs into consideration is worthwhile. We propose HASHNWALK, an incremental algorithm that detects anomalies in a stream of hyperedges. It maintains and updates a constant-size summary of the structural and temporal information about the stream. Using the summary, which is the form of a proximity matrix, HASHNWALK measures the anomalousness of each new hyperedge as it appears. HASHNWALK is **(a) Fast:** it processes each hyperedge in near real-time and billions of hyperedges within a few hours, **(b) Space Efficient**: the size of the maintained summary is a predefined constant, **(c) Effective:** it successfully detects anomalous hyperedges in real-world hypergraphs.

1 Introduction

A variety of real-world graphs, including computer networks, online social networks, and hyperlink networks, have been targets of attacks. Distributed denial-of-service attacks block the availability by causing an unexpected traffic jam on the target machine. In addition, fake connections in online social networks degrade the quality of recommendations, and those in hyperlink networks manipulate the centrality of webpages. Due to its importance and necessity in real-world applications, anomaly detection in graphs has received considerable attention. To detect nodes, edges, and/or subgraphs deviating from structural and temporal patterns in graphs, various numerical measures of the deviation have been proposed with search algorithms [Akoglu *et al.*, 2010; Hooi *et al.*, 2016; Shin *et al.*, 2018]. As many real-world graphs evolve over time, detecting anomalies in real-time, as they appear, is desirable [Bhatia *et al.*, 2020; Eswaran and Faloutsos, 2018].

While graphs model pairwise interactions, interactions in many real-world systems are groupwise (collaborations of co-authors, group interactions on online Q&A platforms, co-purchases of items, etc). Such a groupwise interaction is naturally represented as a *hyperedge*, i.e., a set of an arbitrary number of nodes. A *hypergraph*, which is a set of hyperedges, is an indispensable extension of a graph, which can only describe pairwise relations. Moreover, many of such real-world hypergraphs evolve over time (e.g., emails exchanged continuously between sets of users, co-authorships established over time, and daily records of co-purchased items), and thus they are typically modeled as a stream of hyperedges.

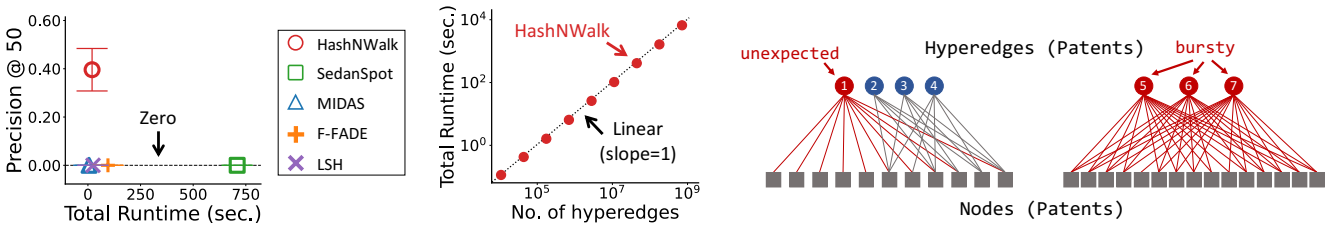
Despite the great interest in anomaly detection in graphs, the same problem in hypergraphs has been largely unexplored. High-order relationships represented by hyperedges exhibit structural and temporal properties distinguished from those in graphs and hence raise unique technical challenges. Thus, instead of simply decomposing hyperedges into pairwise edges and applying graph-based methods, it is required to take the underlying high-order structures into consideration for anomaly detection in hypergraphs.

To this end, we propose HASHNWALK, an online algorithm for detecting anomalous hyperedges. HASHNWALK maintains a constant-size summary that tracks structural and temporal patterns in high-order interactions in the input stream. Specifically, HASHNWALK incorporates so-called *edge-dependent node weights* [Chitra and Raphael, 2019] into random walks on hypergraphs to estimate the proximity between nodes while capturing high-order information. Furthermore, we develop an incremental update scheme, which each hyperedge is processed by as it appears.

The designed hypergraph summary is used to score the anomalousness of any new hyperedge in the stream. While the definition of anomaly depends on the context, in this work, we focus on two intuitive aspects: *unexpectedness* and *burstiness*. We assume that unexpected hyperedges consist of unnatural combinations of nodes, and bursty hyperedges suddenly appear in a short period of time. Based on the information in the form of a hypergraph summary, we formally define two anomaly score metrics that effectively capture these properties. We empirically show that HASHNWALK is effective in detecting anomalous hyperedges in (semi-)real hypergraphs.

In summary, our contributions are as follows:

- **Fast:** It takes a very short time for HASHNWALK to pro-



(a) Speed and preciseness (b) Scalability (c) Effectiveness

Figure 1: Strengths of HASHNWALK. (a) HASHNWALK spots anomalous hyperedges rapidly and precisely in a real-world hypergraph. (b) The total runtime of HASHNWALK is linear in the size of the input hyperedge stream. (c) HASHNWALK detects interesting patents. Patent 1 cited multiple patents that have not been cited together before, and patents 5-7 cited almost the same set of patents. See Section 5 for details.

cess each new hyperedge. Specifically, in our experimental setting, it processed 1.4 billion hyperedges within 3 hours.

- **Space Efficient:** The user can bound the size of the summary, which HASHNWALK maintains.
- **Accurate:** HASHNWALK successfully detects anomalous hyperedges. Numerically, it outperforms its state-of-the-art competitors with up to 47% higher AUROC.

Reproducibility: The source code and datasets are available at https://github.com/geonlee0325/HashNWalk.

2 Related Works

We discuss prior works on the three topics relevant to this paper: (a) anomaly detection in (hyper)graphs; (b) summarization of edge streams; and (c) hypergraphs and applications.

2.1 Anomaly Detection in Graphs & Hypergraphs.

The problem of detecting anomalous nodes, edges, and/or subgraphs has been extensively studied for both static and dynamic graphs [Akoglu *et al.*, 2015]. In static graphs, nodes whose ego-nets are structurally different from others [Akoglu *et al.*, 2010], edges whose removal significantly reduces the encoding cost [Chakrabarti, 2004], or subgraphs whose density is abnormally high [Beutel *et al.*, 2013; Hooi *et al.*, 2016; Shin *et al.*, 2018] are assumed to be anomalies. In dynamic graphs, temporal edges are assumed to be anomalous if they connect sparsely connected parts in graphs [Eswaran and Faloutsos, 2018] or are unlikely to appear according to underlying models [Aggarwal *et al.*, 2011; Yoon *et al.*, 2019; Bhatia *et al.*, 2020]. In addition, dense subgraphs generated within a short time are considered to be anomalous [Shin *et al.*, 2017; Eswaran *et al.*, 2018]. Recently, embedding based methods have shown to be effective in detecting anomalies in graphs [Yu *et al.*, 2018; Chang *et al.*, 2021].

On the other hand, detecting anomalies in hypergraphs is relatively unexplored. Anomalous nodes in the hypergraph have been the targets of detection by using scan statistics on hypergraphs [Park *et al.*, 2009] or training a classifier based on the high-order structural features of the nodes [Leontjeva *et al.*, 2012]. The anomalousness of unseen hyperedges is measured based on how likely the combinations of nodes are drawn from the distribution of anomalous co-occurrences, which is assumed to be uniform, instead of the distribution of nominal ones [Silva and Willett, 2008]. Approximate frequencies of structurally similar hyperedges obtained by locality sensitive hashing are used to score the anomalousness of hyperedges in the hyperedge stream [Ranshous *et al.*, 2017].

In this paper, we compare ours with the methods that detect anomalous interactions in online settings, i.e., anomaly detectors designed for (hyper)edge streams.

2.2 Summarization of Edge Streams.

Summarization aims to reduce the size of a given graph while approximately maintaining its structural properties. It has been particularly demanded in the context of real-time processing of streaming edges. In [Bhatia *et al.*, 2020], a count-min-sketch is maintained for approximate frequencies of edges. Edge frequencies have been used to answer queries regarding structural properties of graphs [Zhao *et al.*, 2011; Tang *et al.*, 2016]. In [Bandyopadhyay *et al.*, 2016], local properties, such as the number of triangles, are estimated by maintaining topological information of a given graph.

2.3 Empirical Analysis of Hypergraphs.

Structural patterns [Benson *et al.*, 2018a; Do *et al.*, 2020; Lee *et al.*, 2020; Lee *et al.*, 2021; Choe *et al.*, 2022] and dynamical patterns [Benson *et al.*, 2018a; Benson *et al.*, 2018b; Kook *et al.*, 2020; Lee and Shin, 2021; Choo and Shin, 2022] in real-world hypergraphs have been studied extensively, and they are useful for finding anomalies that deviate from them.

3 Preliminaries

In this section, we introduce notations and preliminaries.

3.1 Notations and Concepts

Hypergraphs. A *hypergraph* $G = (V, E)$ consists of a set of nodes $V = \{v_1, ..., v_{|V|}\}$ and a set of hyperedges $E = \{e_1, ..., e_{|E|}\}$. Each hyperedge $e \in E$ is a non-empty subset of an arbitrary number of nodes. The *incidence matrix* of G is denoted by $H \in \{0, 1\}^{|E| \times |V|}$, where each entry H_{ij} is 1 if $v_j \in e_i$ and 0 otherwise. A *hyperedge stream* $\{(e_i, t_i)\}_{i=0}^{\infty}$ is a sequence of hyperedges where each hyperedge e_i arrives at time t_i. For any i and j, if $i < j$, then $t_i \leq t_j$.

Clique Expansion and Information Loss. *Clique expansion* [Zhou *et al.*, 2007], where each hyperedge $e \in E$ is converted to a clique composed of the nodes in e, is one of

the most common ways of transforming a hypergraph G into an ordinary pairwise graph. Clique expansion suffers from the loss of information on high-order interactions. That is, in general, a hypergraph is not uniquely identifiable from its clique expansion. Exponentially many non-isomorphic hypergraphs are reduced to identical clique expansions.

Random Walks on Hypergraphs. A random walk on a hypergraph G is formulated in [Chitra and Raphael, 2019] as follows. If the current node is u, **(1)** select a hyperedge e that contains the node u (i.e., $u \in e$) with probability proportional to $\omega(e)$ and **(2)** select a node $v \in e$ with probability proportional to $\gamma_e(v)$ and walk to node v. The weight $\omega(e)$ is the weight of the hyperedge e, and the weight $\gamma_e(v)$ is the weight of node v with respect to the hyperedge e. The weight $\gamma_e(v)$ is *edge-independent* if it is identical for every hyperedge e; and otherwise, it is *edge-dependent*. If all node weights are edge-independent, then a random walk on G becomes equivalent to a random walk on its clique expansion [Chitra and Raphael, 2019]. However, if node weights are edge-dependent, random walks on hypergraphs are generally *irreversible*. That is, they may not be the same as random walks on any undirected graphs. In this sense, if edge-dependent weights are available, random walks are capable of exploiting high-order information beyond clique expansions and thus empirically useful in many machine learning tasks [Hayashi *et al.*, 2020].

Transition Matrix. To incorporate edge-dependent node weights, the incidence matrix H is generalized to a weighted incidence matrix $R \in \mathbb{R}_{\geq 0}^{|E|\times|V|}$ where each entry R_{ij} is $\gamma_{e_i}(v_j)$ if $v_j \in e_i$ and 0 otherwise. Then, the transition probability matrix $P \in \mathbb{R}^{|V|\times|V|}$ of a random walk on the hypergraph G is written as $P = D_V^{-1} W D_E^{-1} R$, where $W \in \mathbb{R}^{|V|\times|E|}$ denotes the hyperedge-weight matrix where each entry W_{ji} is $\omega(e_i)$ if $v_j \in e_i$ and 0 otherwise. The matrices $D_V \in \mathbb{R}^{|V|\times|V|}$ and $D_E \in \mathbb{R}^{|E|\times|E|}$ are diagonal matrices of node degrees and hyperedge weights, respectively. That is, if we let $q \in \mathbb{R}^{|E|}$ and $r \in \mathbb{R}^{|V|}$ be the vectors whose entries are all ones, then $D_V = \text{diag}(Wq)$ and $D_E = \text{diag}(Rr)$.

3.2 Problem Description

The problem that we address in this paper is as follows.

Problem 1. *Given a stream $\{(e_i, t_i)\}_{i=1}^{\infty}$ of hyperedges, detect anomalous hyperedges, whose structural or temporal properties deviate from general patterns, in **near real-time** using **constant space**.*

While the definition of anomalous hyperedges depends on the context, we focus on two intuitive perspectives. In one aspect, a hyperedge is anomalous if it consists of an *unexpected* subset of nodes. That is, we aim to detect hyperedges composed of unusual combinations of nodes. In the other aspect, we aim to identify a set of similar hyperedges that appear *in bursts* as an anomaly. The sudden emergence of similar interactions often indicates malicious behavior harmful in many applications. In addition, for time-critical applications, we aim to detect such anomalous hyperedges in near real-time, as they appear, using bounded space. While one might tempt to reduce hyperedges into subgraphs and solve the problem as anomalous subgraph detection, this harms the high-order information of the hyperedges. Also, existing works on anomalous subgraphs assume static graphs [Hooi *et al.*, 2016] or detect only the single most anomalous subgraph [Shin *et al.*, 2017], while we aim to score every hyperedge in the stream.

4 Proposed Method

In this section, we propose HASHNWALK (Algorithm 1), which is a fast and space-efficient algorithm for detecting anomalies in a hyperedge stream. Our main focus is speed and space efficiency since HASHNWALK is expected to process a potentially infinite stream. As illustrated in Figure 2, it maintains a concise and informative summary of a hyperedge stream (Sect. 4.1), which is incrementally updated as each new hyperedge arrives (Sect. 4.2). Once the summary is updated, anomalous hyperedges are identified immediately based on two principled metrics (Sect. 4.3). While HASHNWALK is based on multiple summaries (Sect. 4.4), we assume that it consists of a single summary for ease of explanation.

4.1 Hypergraph Summarization

Hyperedge Representation. We describe how to concisely represent each hyperedge using constant space. Hyperedges, by definition, are flexible in their sizes, and it is non-trivial to represent each hyperedge using the same amount of space. To this end, we map each node into one of M different values using a hash function $h(\cdot) : V \to \{1, ..., M\}$. We consider each hash value as a *supernode* that contains the nodes with the same hash value. Due to hash collisions, a hyperedge may contain a supernode multiple times, and the number of occurrences becomes the weight of the supernode with respect to the hyperedge. Formally, we represent each hyperedge e of *any size* into a M-dimensional vector $m(e) \in \mathbb{Z}^M$, whose k^{th} element indicates the number of the nodes that are contained in e and mapped into the hash value k (i.e., $m_k(e) := \sum_{v \in e} \mathbb{1}(h(v) = k)$). It is also interpreted as the weight of the supernode k with respect to the hyperedge e. We denote $\tilde{e}$ as the set of supernodes that hyperedge e contains, i.e., $\tilde{e} := \{k \,|\, m_k(e) > 0\}$. Note that a hyperedge of any size is represented as a fixed-size vector, whose size M is user-controlled. In addition, the edge-dependent weights of supernodes can be utilized by random walks (see Section 3.1). If we use a constant-time hash function h and a sparse vector format, for each hyperedge e, the time complexity of generating the vector $m(e)$ is $O(|e|)$, as stated in Lemma 1.

Lemma 1 (Time Complexity of Generating $m(e)$)**.** *Given a hyperedge e, it takes $O(|e|)$ time to generate the vector $m(e)$.*

PROOF. *Creating a zero vector in a sparse format (e.g., a hash table) and incrementing $m_{h(v)}(e)$ for every node $v \in e$ takes $O(|e|)$ time.* □

Hypergraph Summary. Below, we describe how to summarize the entire hypergraph for rapid and accurate anomaly detection. We note that the key building block for identifying anomalous hyperedges of both types (i.e., unexpected ones and similar ones in bursts) is to estimate the proximity or structural similarity between nodes. Thus, we summarize

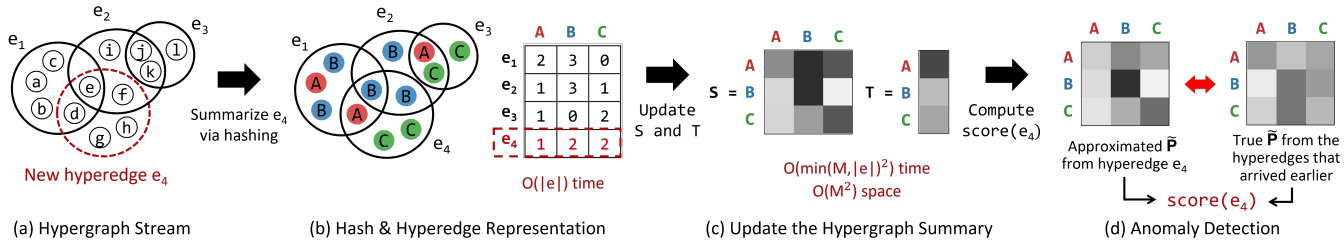


Figure 2: Outline of HASHNWALK. (a) A new hyperedge arrives in the input hyperedge stream. (b) Nodes are merged into M supernodes with edge-dependent weights by hashing, and hyperedges, including the new one, are represented as M-dimensional vectors (M=3 in this example). (c) The hypergraph summary is composed of a matrix S and a vector T, and it is incrementally updated in response to the new hyperedge. (d) Based on the summary $\tilde{P}$, which is immediately obtainable from S and T (Eq. (2)), the anomalousness of the new hyperedge is measured using the proposed scoring functions (Eq. (5)).

Algorithm 1 HASHNWALK

Input: **(1)** hyperedge stream: $\mathcal{E} = \{(e_i, t_i)\}_{i=1}^{\infty}$, **(2)** number of supernodes M, **(3)** number of hash functions K, **(4)** time-decaying parameter α
Output: stream of anomaly scores $\{y_i\}_{i=1}^{\infty}$

```
S ∈ ℝ^{M×M} and T ∈ ℝ^M                          ▷ Initialize to zeros
for each hyperedge (e_i, t_i) ∈ ℰ do
    m(e_i) ← summarize e_i via hashing            ▷ Sect. 4.1
    update S and T                                ▷ Sect. 4.2
    y_i ← (score_U(e_i), score_B(e_i))            ▷ Sect. 4.3
end for
return {y_i}_{i=1}^{∞}
```

the input hypergraph in the form of proximity between supernodes, and we extend random walks to measure the proximity. Our summary is based on random walks extended with edge-dependent supernode weights and hyperedge weights; and we use the transition probabilities as their approximation for rapid updates (see Section 4.2). Specifically, we summarize the input hypergraph as a matrix $\tilde{P} := \tilde{D}_V^{-1}\tilde{W}\tilde{D}_E^{-1}\tilde{R} \in \mathbb{R}^{M\times M}$, where $\tilde{R} \in \mathbb{R}^{|E|\times M}$ is the weighted incidence matrix where each entry $\tilde{R}_{i\tilde{v}}$ is $\gamma_{\tilde{e}}(\tilde{v})$ if $\tilde{v} \in \tilde{e}_i$ and 0 otherwise. The matrix $\tilde{W} \in \mathbb{R}^{M\times|E|}$ denotes the hyperedge-weight matrix where $\tilde{W}_{\tilde{v}\tilde{e}}$ is $\omega(\tilde{e})$ if $\tilde{v} \in \tilde{e}$ and 0 otherwise. The matrices $\tilde{D}_V \in \mathbb{R}^{M\times M}$ and $\tilde{D}_E \in \mathbb{R}^{|E|\times|E|}$ are diagonal matrices of supernode degrees and hyperedge weights, respectively. Then, $\tilde{P}$ is the transition probability matrix where each entry $\tilde{P}_{\tilde{u}\tilde{v}}$ is the transition probability from supernode $\tilde{u}$ to $\tilde{v}$:

$$\tilde{P}_{\tilde{u}\tilde{v}} = \sum_{i=1}^{|E|} \frac{\omega(\tilde{e}_i)\cdot\mathbb{1}(\tilde{u}\in\tilde{e}_i)}{\tilde{W}_{\tilde{u}}} \cdot \frac{\gamma_{\tilde{e}_i}(\tilde{v})}{\tilde{R}_{\tilde{e}_i}} \tag{1}$$

where $\tilde{W}_{\tilde{u}}$ is the weighted degree of the $\tilde{u}$, i.e., $\tilde{W}_{\tilde{u}} := \sum_{i=1}^{|E|}\omega(\tilde{e}_i)\cdot\mathbb{1}(\tilde{u}\in\tilde{e}_i)$, and $\tilde{R}_{\tilde{e}_i}$ is the sum of the weights of the supernodes in the $\tilde{e}_i$, i.e., $\tilde{R}_{\tilde{e}_i} := \sum_{\tilde{v}\in\tilde{e}_i}\gamma_{\tilde{e}_i}(\tilde{v})$.

Edge-Dependent Supernode Weights. If edge-dependent supernode weights are available, random walks utilize high-order information beyond clique expansions. Such weights are naturally obtained from the aforementioned vector representation of hyperedges. That is, we use the number of the occurrences of each supernode $\tilde{v}$ in each hyperedge $\tilde{e}_i$ as the weight of $\tilde{v}$ with respect to $\tilde{e}_i$. Formally, $\gamma_{\tilde{e}_i}(\tilde{v}) = m_{\tilde{v}}(e_i)$, and thus $R_{\tilde{e}_i} = \sum_{\tilde{v}\in\tilde{e}_i}\gamma_{\tilde{e}_i}(\tilde{v}) = \sum_{k=1}^{M} m_k(e_i) = |e_i|$.

Time-Decaying Hyperedge Weights. In order to facilitate identifying recent bursts of similar hyperedges, which are one of our focuses, we emphasize recent hyperedges with large weights. Specifically, at current time t, we define the weight of each hyperedge e_i, which is arrived at time t_i, as $\omega(e_i) = \ker(t - t_i) = \alpha^{t-t_i}(1-\alpha)$ where $\ker(x) := \alpha^x(1-\alpha)$ is a kernel function for quantifying time decay and $\alpha \in [0, 1)$ is a hyperparameter that determines the degree of emphasis. Specifically, smaller α more emphasizes recent hyperedges.

4.2 Incremental Update

Challenges. Constructing $\tilde{P}$ from scratch, which takes $O(|E|\cdot M^2)$ time, is undesirable when immediate responses to anomalies are demanded. In addition, when hyperedges are streamed indefinitely, materializing $\tilde{W}$, $\tilde{D}_E$, and $\tilde{R}$, which are used to compute $\tilde{P}$, is prohibitive since their sizes are proportional to the number of hyperedges.

Proposed Updated Scheme. We present an incremental algorithm for efficiently but exactly updating $\tilde{P}$ in response to a new hyperedge. The proposed update scheme maintains only $\tilde{P}$, whose size is controllable by the user, without materializing any larger matrix. Assume m hyperedges $e_1, ..., e_m$ have arrived, and let $\tilde{P}^{(m)}_{\tilde{u}\tilde{v}}$ be the proximity from supernode $\tilde{u}$ to supernode $\tilde{v}$ in them. We introduce a matrix $S \in \mathbb{R}^{M\times M}$ and a vector $T \in \mathbb{R}^M$, and for any supernodes $\tilde{u}$ and $\tilde{v}$, their entries when the hyperedge e_m arrives at time t_m are

$$S^{(m)}_{\tilde{u}\tilde{v}} := \sum_{i=1}^{m} \alpha^{-t_i}\cdot\mathbb{1}(\tilde{u}\in\tilde{e}_i)\cdot\frac{\gamma_{\tilde{e}_i}(\tilde{v})}{\tilde{R}_{\tilde{e}_i}}$$

$$T^{(m)}_{\tilde{u}} := \sum_{i=1}^{m} \alpha^{-t_i}\cdot\mathbb{1}(\tilde{u}\in\tilde{e}_i),$$

Then, based on Eq. (1) and the predefined hyperedge weight function $\ker(x) = \alpha^x(1-\alpha)$, $\tilde{P}^{(m)}_{\tilde{u}\tilde{v}}$ is written as

$$\tilde{P}^{(m)}_{\tilde{u}\tilde{v}} = \frac{\sum_{i=1}^{m}\alpha^{t_m-t_i}(1-\alpha)\cdot\mathbb{1}(\tilde{u}\in\tilde{e}_i)\cdot\frac{\gamma_{\tilde{e}_i}(\tilde{v})}{\tilde{R}_{\tilde{e}_i}}}{\sum_{i=1}^{m}\alpha^{t_m-t_i}(1-\alpha)\cdot\mathbb{1}(\tilde{u}\in\tilde{e}_i)} = \frac{S^{(m)}_{\tilde{u}\tilde{v}}}{T^{(m)}_{\tilde{u}}}. \tag{2}$$

Instead of directly tracking the proximity matrix $\tilde{P}$, we track aforementioned S and T, whose entries are initialized to zero. Each entry $S_{\tilde{u}\tilde{v}}$ and $T_{\tilde{u}}$ can be updated in constant time, as presented in Lemmas 2 and 3, and once they are updated, we can compute $\tilde{P}^{(m)}_{\tilde{u}\tilde{v}}$ in $O(1)$ time by Eq. (2), if necessary.

Lemma 2 (Updating $S_{\tilde{u}\tilde{v}}$)**.** *For any $m \geq 0$, when the hyperedge e_{m+1} arrives at t_{m+1}, Eq.* (3) *holds.*

$$S^{(m+1)}_{\tilde{u}\tilde{v}} = S^{(m)}_{\tilde{u}\tilde{v}} + \alpha^{-t_{m+1}} \cdot \mathbb{1}(\tilde{u} \in \tilde{e}_{m+1}) \cdot \frac{\gamma_{\tilde{e}_{m+1}}(\tilde{v})}{\tilde{R}_{\tilde{e}_{m+1}}}. \quad (3)$$

Lemma 3 (Updating $T_{\tilde{u}}$)**.** *For any $m \geq 0$, when the hyperedge e_{m+1} arrives at t_{m+1}, Eq.* (4) *holds.*

$$T^{(m+1)}_{\tilde{u}} = T^{(m)}_{\tilde{u}} + \alpha^{-t_{m+1}} \cdot \mathbb{1}(\tilde{u} \in \tilde{e}_{m+1}). \quad (4)$$

Lemma 2 and Lemma 3 are immediate from the definitions of $S^{(m)}_{\tilde{u}\tilde{v}}$ and $T^{(m)}_{\tilde{u}}$.

Complexity. Notably, if $\tilde{u} \notin \tilde{e}_{m+1}$, $\mathbb{1}(\tilde{u} \in \tilde{e}_{m+1}) = 0$ holds and if $\tilde{v} \notin \tilde{e}_{m+1}$, $\gamma_{\tilde{e}_{m+1}}(\tilde{v}) = 0$ holds. Thus, if $\tilde{u}$ or $\tilde{v}$ is not included in the new hyperedge (i.e., $\tilde{u} \notin \tilde{e}_{m+1}$ or $\tilde{v} \notin \tilde{e}_{m+1}$), $S_{\tilde{u}\tilde{v}}$ remains the same (i.e., $S^{(m+1)}_{\tilde{u}\tilde{v}} = S^{(m)}_{\tilde{u}\tilde{v}}$) and thus does not need any update. Similarly, $T_{\tilde{u}}$ does not change if $\tilde{u}$ is not included in $\tilde{e}_{m+1}$. These facts significantly reduce the update time of the summary, enabling near real-time processing of each hyperedge. To sum up, in response to a new hyperedge, HASHNWALK updates the summary in a short time using constant space, as stated in Lemmas 4 and 5, respectively.

Lemma 4 (Update Time Per Hyperedge)**.** *Given the sparse vector representation $m(e)$ of a hyperedge e, updating $S \in \mathbb{R}^{M \times M}$ and $T \in \mathbb{R}^{M}$ using Eq.* (3) *and Eq.* (4) *takes $O(\min(M, |e|)^2)$ time.*

PROOF. *The number of supernodes in e is $|\tilde{e}|$, which is at most the number of nodes $|e|$ and the number of supernodes M, and thus $|\tilde{e}| = O(\min(M, |e|))$. Then, $|\tilde{e}|^2$ elements of S and $|\tilde{e}|$ elements of T are updated by Eq.* (3) *and Eq.* (4)*, and the update time is constant per element. Therefore, the total time complexity is $O(\min(M, |e|)^2)$.* ■

Lemma 5 (Constant Space)**.** *The maintained summary $\tilde{P}$ takes $O(M^2)$ space.*

PROOF. *The matrix S and the vector T require $O(M^2)$ and $O(M)$ space, respectively.* ■

4.3 Anomaly Detection

Hyperedge Anomaly Score. We now propose an online anomalous hyperedge detector, which is based on the structural and temporal information captured in the summary $\tilde{P}$. We evaluate each newly arriving hyperedge by measuring a hyperedge anomaly score defined in Definition 1.

Definition 1 (Hyperedge Anomaly Score)**.** *Given a newly arriving hyperedge e_i at time t_i, its anomaly score is defined as*

$$\mathsf{score}(e_i) = \underset{\tilde{u},\tilde{v} \in \tilde{e}_i}{aggregate} \left(d^{\beta}_{\tilde{u},t_i} \cdot \log \frac{a_{\tilde{u}\tilde{v}}}{s_{\tilde{u}\tilde{v}}} \right), \quad (5)$$

where $d_{\tilde{u},t_i}$ is the number of occurrences of $\tilde{u}$ at time t_i, $\beta \in [0, \infty)$ is a hyperparameter for the importance of the occurrences, $a_{\tilde{u},\tilde{v}} = \frac{\gamma_{\tilde{e}_i}(\tilde{v})}{\tilde{R}_{\tilde{e}_i}}$, and $s_{\tilde{u},\tilde{v}}$ is $\tilde{P}_{\tilde{u},\tilde{v}}$ just before t_i. Intuitively, $a_{\tilde{u},\tilde{v}}$ and $s_{\tilde{u},\tilde{v}}$ are the "observed" proximity (i.e., the proximity in the current hyperedge) and "expected" proximity (i.e., the proximity in all past hyperedges appearing before t_i) from supernode $\tilde{u}$ to supernode $\tilde{v}$, respectively.

Note that the relationships between all pairs of supernodes in the hyperedge, including the pairs of the same supernode, are taken into consideration, and they are aggregated using any aggregation functions. The hyperparameter β and the aggregate function can be controlled to capture various types of anomalies. For the two types of anomalies, we define scoring functions $\mathsf{score}_\mathsf{U}$ and $\mathsf{score}_\mathsf{B}$ as described below.

Unexpectedness ($\mathsf{score}_\mathsf{U}$). Intuitively, $a_{\tilde{u},\tilde{v}}/s_{\tilde{u},\tilde{v}}$ in Eq. (5) measures how much the proximity from the supernode $\tilde{u}$ to $\tilde{v}$ in the new hyperedge e_i deviates from the proximity in the past hyperedges. Specifically, the ratio is high if two supernodes $\tilde{u}$ and $\tilde{v}$ that have been far from each other in past hyperedges unexpectedly co-appear with high proximity in the new hyperedge. Thus, in $\mathsf{score}_\mathsf{U}$, which is the anomaly score for identifying unexpected hyperedges, we focus on the ratio by setting $\beta = 0$. In order to detect any such unexpected pairs of supernodes in the hyperedge, $\mathsf{score}_\mathsf{U}$ uses the maximum ratio as the final score (i.e., $\mathsf{aggregate} = \mathsf{max}$).

Burstiness ($\mathsf{score}_\mathsf{B}$). In order to detect similar hyperedges that appear in bursts, the number of occurrences of supernodes is taken into consideration. Supernodes, by definition, are subsets of nodes, and similar hyperedges tend to share many supernodes. If a large number of similar hyperedges appear in a short period of time, then the occurrences of the supernodes in them tend to increase accordingly. Thus, in $\mathsf{score}_\mathsf{B}$, which is the anomaly score for identifying recent bursts of similar hyperedges, we set β to a positive number (specifically, 1 in this work) to take such occurrences (i.e., $d^{\beta}_{\tilde{u},t_i}$ in Eq. (5)) into consideration, in addition to unexpectedness (i.e., $a_{\tilde{u},\tilde{v}}/s_{\tilde{u},\tilde{v}}$ in Eq. (5)). We reflect the degrees of all supernodes in the hyperedge by averaging the scores from all supernode pairs (i.e., $\mathsf{aggregate} = \mathsf{mean}$).

Complementarity of the Anomaly Scores. While the only differences between $\mathsf{score}_\mathsf{U}$ and $\mathsf{score}_\mathsf{B}$ are the consideration of the current degree of supernodes (i.e., $d^{\beta}_{\tilde{u},t_i}$) and the aggregation methods, the differences play an important role in identifying specific types of anomalies (see Section 5.2).

Complexity. For each new hyperedge e, HASHNWALK computes $\mathsf{score}(e)$ in a short time, as stated in Lemma 6.

Lemma 6 (Scoring Time Per Hyperedge)**.** *Given the hypergraph summary $\tilde{P}$ and a hyperedge e in the form of a vector $m(e)$, computing $\mathsf{score}(e)$ takes $O(\min(M, |e|)^2)$ time.*

PROOF. *The number of supernodes in e is $O(\min(M, |e|))$. We maintain and update the current degrees of supernodes, which takes $O(\min(M, |e|))$ time for each new hyperedge e. There are $O(\min(M, |e|)^2)$ pairs of supernodes in $\tilde{e}$, and the computation for each supernode pair in Eq.* (5) *takes $O(1)$ time. Hence, the total time complexity is $O(\min(M, |e|)^2)$.* ■

Theorem 1 (Total Time Per Hyperedge)**.** HASHNWALK *takes $O(|e| + \min(M, |e|)^2)$ time to process a hyperedge e.*

PROOF. *Theorem 1 follows from Lemmas 1, 4, and 6.* ■

Dataset	$\lvert\mathbf{V}\rvert$	$\lvert\mathbf{E}\rvert$	$\text{avg}_{\mathbf{e}\in\mathbf{E}}\lvert\mathbf{e}\rvert$	$\text{max}_{\mathbf{e}\in\mathbf{E}}\lvert\mathbf{e}\rvert$
Email-Enron	143	10,885	2.472	37
Transaction	284,807	284,807	5.99	6
DBLP	1,930,378	3,700,681	2.790	280
Cite-patent	4,641,021	1,696,554	18.103	2,076
Tags-overflow	49,998	14,458,875	2.968	5

Table 1: Five real-world hypergraphs.

4.4 Using Multiple Summaries (Optional)

Multiple hash functions can be used in HASHNWALK to improve its accuracy at the expense of speed and space. Specifically, if we use K hash functions, maintain K summaries, and compute K scores independently, then the space and time complexities become K times of those with one hash function. Given hyperedge anomaly scores from K different summaries, we use the **maximum** one as the final score, although any other aggregation function can be used instead.

5 Experiments

We review our experiments to answer Q1-Q4:

Q1. **Performance:** How rapidly and accurately does HASHNWALK detect anomalous hyperedges?

Q2. **Discovery:** What meaningful events can HASHNWALK detect in real-world hypergraph streams?

Q3. **Scalability:** How does the total runtime of HASHNWALK change with respect to the input stream size?

Q4. **Parameter Analysis:** How do the parameters of HASHNWALK affect its performance?

5.1 Experimental Settings

Datasets. We used five different real-world datasets in Table 1. They are described in detail in later subsections.

Machines. We ran F-FADE on a workstation with an Intel Xeon 4210 CPU, 256GB RAM, and RTX2080Ti GPUs. We ran the others on a desktop with an Intel Core i9-10900KF CPU and 64GB RAM.

Baselines. We consider four streaming algorithms for anomaly detection in graphs and hypergraphs as competitors:

- **SEDANSPOT [Eswaran and Faloutsos, 2018]:** Given a stream of edges, it aims to detect *unexpected edges*, i.e., edges that connect nodes from sparsely connected parts of the graph, based on personalized PageRank scores.
- **MIDAS [Bhatia *et al.*, 2020]:** Given a stream of edges, it aims to detect *similar edges in bursts*. To this end, it uses the Count-Min-Sketch.
- **F-FADE [Chang *et al.*, 2021]:** Given a stream of edges, it uses frequency-based matrix factorization and computes the likelihood-based anomaly score of each edge that combines *unexpectedness* and *burstiness*.
- **LSH [Ranshous *et al.*, 2017]:** Given a stream of hyperedges, it computes the *unexpectedness* of each one using its approximate frequency so far.

For graph-based anomaly detection methods, we transform hypergraphs into graphs via clique expansion (Section 3.1). That is, each hyperedge e_i is reduced to $|e_i|^2$ pairwise edges, and the timestamp t_i is assigned to each edge. The anomaly score of the hyperedge is computed by aggregating the anomaly scores of the pairwise edges, using the best one among arithmetic/geometric mean, sum, and maximum.

Implementation. We implemented HASHNWALK and LSH in C++ and Python, respectively. For the others, we used the official open-source implementation.

Evaluation. Given anomaly scores of hyperedges, we measure AUROC and Precision@k (i.e., the ratio of true positives among k hyperedges with the highest scores).

5.2 Q1. Performance Comparison

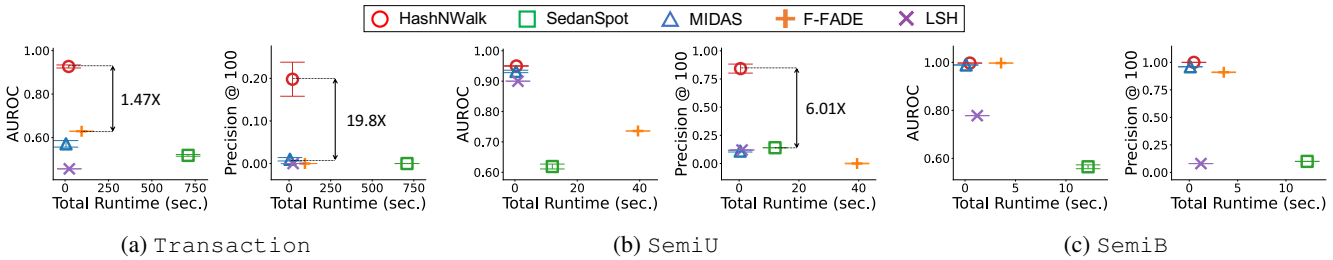
We consider three hypergraphs: `Transaction`, `SemiU`, and `SemiB`. `Transaction` [Dal Pozzolo *et al.*, 2015] is a real-world hypergraph of credit card transactions. Each timestamped transaction is described by a 28 dimensional feature vector. There exist 492 frauds, which account for 0.172% of the entire transactions. For each transaction, we generate a hyperedge by grouping it with 5 nearest transactions that occurred previously. Thus, each node is a transaction, and each hyperedge is a set of transactions that are similar to each other.

In `Email-Enron`, each node is an email account and each hyperedge is the set of the sender and receivers. The timestamp of each hyperedge is when the email was sent. We consider two scenarios INJECTIONU and INJECTIONB, where we generate two semi-real hypergraphs `SemiU` and `SemiB` by injecting 200 unexpected and bursty hyperedges, respectively, in `Email-Enron`. The two injection scenarios are designed as follows:

- **INJECTIONU**: Injecting unexpected hyperedges.
  1. Select a hyperedge $(e_i, t_i) \in \mathcal{E}$ uniformly at random.
  2. Create a hyperedge by replacing $\lceil |e_i| / 2 \rceil$ nodes in e_i with random ones, and set their timestamp to t_i.
  3. Repeat (1)-(2) g times to generate g hyperedges.
- **INJECTIONB**: Injecting bursty hyperedges.
  1. Select a time $t \in \{t_{\text{setup}} + 1, \cdots, t_m\}$ uniformly at random.
  2. Sample a set of n nodes $N \subseteq V$ uniformly at random.
  3. Create m uniform random subsets of N at time t. Their sizes are chosen uniformly at random from $\{1, \cdots, n\}$.
  4. Repeat (1) - (3) l times to generate $m \cdot l$ hyperedges.

All anomalies are injected after time t_{setup}, and thus all methods are evaluated from time t_{setup} where we set $t_{\text{setup}} = t_{100}$ ($< t_{|E|\cdot 0.01}$). In INJECTIONU, we set $g = 200$. In INJECTIONB, we set $m = 20$, $n = 5$, and $l = 10$.

Accuracy. In `Transaction`, we use score_{U} and set $\alpha = 0.98$, $K = 4$, and $M = 350$. In `SemiU` and `SemiB`, we use score_{U} and score_{B}, respectively, and commonly set $\alpha = 0.98$, $K = 15$, and $M = 20$. As discussed later, these summaries take up less space than the original hypergraphs. As shown in Figure 3, HASHNWALK accurately detects anomalous hyperedges in real and semi-real hypergraphs. Notably, while most



(a) Transaction (b) SemiU (c) SemiB

Figure 3: HASHNWALK is accurate (in terms of AUROC and Prec.@100) and fast. For example, in the Transaction dataset, HASHNWALK achieves 47% higher AUROC with 4.7× faster speed, compared to F-FADE.

	SemiU		SemiB	
	AUROC	Prec.@100	AUROC	Prec.@100
score_U	**0.951**	**0.815**	0.802	0.740
score_B	0.916	0.090	**0.997**	**1.000**

Table 2: The two proposed hyperedge anomaly scoring metrics score_U and score_B complement each other.

methods fail to find any anomalous hyperedges in their top 50 (see Figure 1a in Section 1) or top 100 (Figure 3a) hyperedges with the highest anomaly scores, HASHNWALK is successful. In addition, HASHNWALK accurately detects both unexpected and bursty hyperedges. Note that while several competitors successfully spot bursty hyperedges in SemiB, most of them fail to spot unexpected ones in SemiU. Specifically, in SemiU, HASHNWALK achieves 6.01× higher precision@100 with 27× faster speed than SEDANSPOT.

Speed. As seen in Figure 3, HASHNWALK is one of the fastest methods among the considered ones. Notably, in SemiU, HASHNWALK is 27× faster than the second most accurate method.

Space Usage. We analyze the amount of space used by HASHNWALK. Let C_Z and C_F be the numbers of bits to encode an integer and a floating number, respectively, and we assume $C_Z = C_F = 32$. The size of the original hypergraph $G = (V, E)$ is the sum of the hyperedge sizes, and precisely, $C_Z \cdot \sum_{e \in E} |e|$ bits are required to encode the hypergraph. As described in Lemma 5 in Section 4.2, for each hash function, HASHNWALK tracks a matrix $S \in \mathbb{R}^{M \times M}$ and a vector $T \in \mathbb{R}^M$, and thus it requires $C_F \cdot K \cdot (M^2 + M)$ bits with K hash functions. We set $K = 4$ and $M = 350$ in Transaction; and $K = 15$ and $M = 20$ in SemiU and SemiB. As a result, HASHNWALK requires about 28.6% and 22.5% of the space required for the original hypergraphs, in Transaction and semi-real hypergraphs, respectively. For competitors, we conduct hyperparameter tuning, including configurations requiring more space than ours.[1]

Complementarity of score_U and score_B. As seen in Table 2, while score_U is effective in detecting unexpected hyperedges, it shows relatively low accuracy in detecting bursty hyperedges. The opposite holds in score_B. These indicate the two metrics score_U and score_B are complementary.

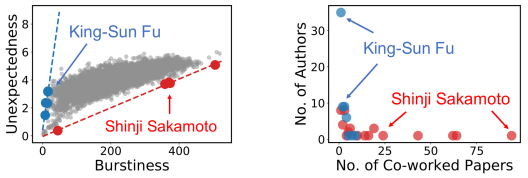
[1] See https://github.com/geonlee0325/HashNWalk for details.

5.3 Q2. Discovery

Here, we share the results of case studies conducted on the DBLP, Cite-patent, and Tags-overflow datasets.

Discoveries in Co-authorship Hypergraph. DBLP contains information of bibliographies of computer science publications. Each node represents an author, and each hyperedge consists of authors of a publication. The timestamp of the hyperedge is the year of publication. Here, we investigate how authors co-work with different researchers. For each author v who have published at least 100 papers, we compute the average unexpectedness and burstiness scores of the hyperedges that v is contained in, which we denote by $\text{avg}_\text{U}(v)$ and $\text{avg}_\text{B}(v)$, respectively. We analyze several authors whose ratio $\text{avg}_\text{U}(v)/\text{avg}_\text{B}(v)$ is the highest or the lowest. Intuitively, authors with low ratios tend to co-work in a bursty manner with expected co-authors, while those with high ratios tend to work steadily with unexpected co-authors. Surprisingly, Dr. Bill Hancock, whose $\text{avg}_\text{U}(v)/\text{avg}_\text{B}(v)$ ratio is the lowest, published 186 papers all alone. Furthermore, Dr. Hancock published 139 papers in 2000. On the other hand, Dr. Seymour Ginsburg, whose $\text{avg}_\text{U}(v)/\text{avg}_\text{B}(v)$ is the highest, published 114 papers from 1958 to 1999 (2.7 papers per year). In addition, 18 co-authors (out of 38) co-authored only one paper with Dr. Ginsburg. In fact, avg_U and avg_B of the most authors are clustered as seen in Figure 4a, and the top authors are those with the largest (or the smallest) slope. We further conduct case studies on two specific authors Dr. Shinji Sakamoto and Dr. King-Sun Fu, whose co-working patterns are very different. As seen in Figure 4b, while Dr. Sakamoto collaborated on most papers with a few researchers, Dr. Fu enjoyed co-working with many new researchers. These findings support our intuition behind the proposed measures, score_U and score_B.

Discoveries in Patent Citation Hypergraph. We use Cite-patent [Tang *et al.*, 2012], which is a citation hypergraph where each node is a patent and each hyperedge is the set of patents cited by a patent. The timestamp of each hyperedge is the year of the citation ranging from 2000 to 2012. Using HASHNWALK, we extract some hyperedges with high score_U or score_B. Then, we represent each hyperedge as a $|V|$-dimensional binary vector indicating which nodes belong to the hyperedge. We visualize the hyperedges after reducing the dimension of the vectors via T-SNE in Figure 5. While unexpected hyperedges are spread, bursty hyperedges are closely located, indicating structurally similar



(a) avg_U and avg_B (b) Two researchers

Figure 4: Case studies on the DBLP dataset. Some authors deviate from the general pattern (4a). Dr. Fu and Dr. Sakamoto differ in their co-working patterns (4b).

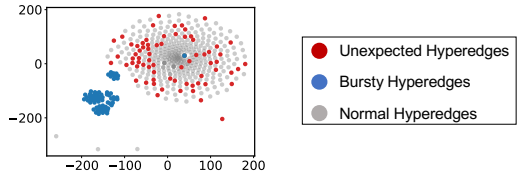


Figure 5: Case study on the Cite-patent dataset. Unexpected & bursty hyperedges have different properties.

hyperedges arrive in bursts. In addition, we closely examine the citation patterns of suspicious patents detected by HASHNWALK. As seen in Figure 1c, patents with unexpected or bursty citations are effectively detected.

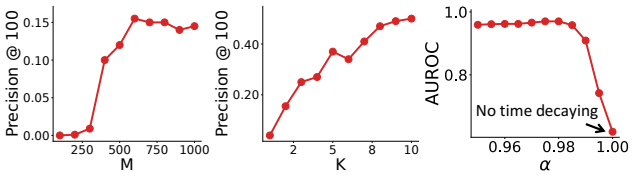
Discoveries in Online Q&A Cite. We share the results of a case study using Tags-overflow. In the dataset, nodes are tags and hyperedges are the set of tags attached to a question. Hyperedges with high score_U (i.e., sets of unexpected keywords) include: {*channel, ignore, antlr, hidden, whitespace*}, {*sifr, glyph, sting, text-styling, embedding*}, and {*retro-computing, boot, floppy, amiga*}. Hyperedges with high score_B (i.e., sets of bursty keywords) include: {*python, javascript*}, {*java, adobe, javascript*}, and {*c#, java*}. Notably, sets of unpopular tags tend to have high unexpectedness, while those containing popular keywords, such as *python* and *javascript*, have high burstiness.

5.4 Q3. Scalability

To evaluate the scalability of HASHNWALK, we measure how rapidly it updates the hypergraph summary and computes the anomaly scores as the number of hyperedges grows. To this end, we upscale Email-Enron, which originally consists of 10,885 hyperedges, by 2^1 to 2^{17} times, and measure the total runtime of HASHNWALK. As seen in Figure 1b, the total runtime is linear in the number of hyperedges, which is consistent with our theoretical analysis (Theorem 1 in Section 4). That is, the time taken for processing each hyperedge is near constant. Notably, HASHNWALK is scalable enough to process a stream of 1.4 billion hyperedges within 3 hours.

5.5 Q4. Parameter Analysis

We evaluate HASHNWALK under different parameter settings, and the results are shown in Figure 6. In most cases, there is a positive correlation with M (i.e., the number of supernodes) and K (i.e., the number of hash functions). Intuitively, a larger number of supernodes and hash functions collectively reduce the variance due to randomness introduced by hash functions. However, since the space usage is dependent on these parameters, a trade-off between the accuracy and the space usage should be considered. In addition, properly setting α (i.e., time decaying parameter) improves the accuracy, as shown in Figure 6, which indicates that not only structural information but also the temporal information is critical in detecting anomalies in hyperedge streams.



(a) Effect of M (b) Effect of K (c) Effect of α

Figure 6: The performance of HASHNWALK depends on the number of supernodes (M), the number of hash functions (K), and time decaying parameter α in the Transaction dataset.

6 Conclusion

In this work, we propose HASHNWALK, an online anomaly detector for hyperedge streams. HASHNWALK maintains a random-walk-based hypergraph summary with constant space, and it is incrementally updated in near real-time. Using the summary, HASHNWALK computes two anomaly scores that are effective in identifying (a) hyperedges composed of unexpected combinations of nodes and (b) those appearing in bursts. Our experiments demonstrate the speed, accuracy, and effectiveness of HASHNWALK in (semi-)real datasets.

Acknowledgments

This work was supported by National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2020R1C1C1008296) 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)).

References

[Aggarwal *et al.*, 2011] Charu C Aggarwal, Yuchen Zhao, and S Yu Philip. Outlier detection in graph streams. In *ICDE*, 2011.

[Akoglu *et al.*, 2010] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *PAKDD*, 2010.

[Akoglu *et al.*, 2015] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *DMKD*, 29(3):626–688, 2015.

[Bandyopadhyay *et al.*, 2016] Bortik Bandyopadhyay, David Fuhry, Aniket Chakrabarti, and Srinivasan Parthasarathy. Topological graph sketching for incremental and scalable analytics. In *CIKM*, 2016.

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

[Benson *et al.*, 2018b] Austin R Benson, Ravi Kumar, and Andrew Tomkins. Sequences of sets. In *KDD*, 2018.

[Beutel *et al.*, 2013] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *WWW*, 2013.

[Bhatia *et al.*, 2020] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. Midas: Microcluster-based detector of anomalies in edge streams. In *AAAI*, 2020.

[Chakrabarti, 2004] Deepayan Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *PKDD*, 2004.

[Chang *et al.*, 2021] Yen-Yu Chang, Pan Li, Rok Sosic, MH Afifi, Marco Schweighauser, and Jure Leskovec. F-fade: Frequency factorization for anomaly detection in edge streams. In *WSDM*, 2021.

[Chitra and Raphael, 2019] Uthsav Chitra and Benjamin Raphael. Random walks on hypergraphs with edge-dependent vertex weights. In *ICML*, 2019.

[Choe *et al.*, 2022] Minyoung Choe, Jaemin Yoo, Geon Lee, Woonsung Baek, U Kang, and Kijung Shin. Midas: Representative sampling from real-world hypergraphs. In *WWW*, 2022.

[Choo and Shin, 2022] Hyunjin Choo and Kijung Shin. On the persistence of higher-order interactions in real-world hypergraphs. In *SDM*, 2022.

[Dal Pozzolo *et al.*, 2015] Andrea Dal Pozzolo, Olivier Caelen, Reid A Johnson, and Gianluca Bontempi. Calibrating probability with undersampling for unbalanced classification. In *SSCI*, 2015.

[Do *et al.*, 2020] Manh Tuan Do, Se-eun Yoon, Bryan Hooi, and Kijung Shin. Structural patterns and generative models of real-world hypergraphs. In *KDD*, 2020.

[Eswaran and Faloutsos, 2018] Dhivya Eswaran and Christos Faloutsos. Sedanspot: Detecting anomalies in edge streams. In *ICDM*, 2018.

[Eswaran *et al.*, 2018] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. Spotlight: Detecting anomalies in streaming graphs. In *KDD*, 2018.

[Hayashi *et al.*, 2020] Koby Hayashi, Sinan G Aksoy, Cheong Hee Park, and Haesun Park. Hypergraph random walks, laplacians, and clustering. In *CIKM*, 2020.

[Hooi *et al.*, 2016] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. Fraudar: Bounding graph fraud in the face of camouflage. In *KDD*, 2016.

[Kook *et al.*, 2020] Yunbum Kook, Jihoon Ko, and Kijung Shin. Evolution of real-world hypergraphs: Patterns and models without oracles. In *ICDM*, 2020.

[Lee and Shin, 2021] Geon Lee and Kijung Shin. Thyme+: Temporal hypergraph motifs and fast algorithms for exact counting. In *ICDM*, 2021.

[Lee *et al.*, 2020] Geon Lee, Jihoon Ko, and Kijung Shin. Hypergraph motifs: Concepts, algorithms, and discoveries. *PVLDB*, 13(11):2256–2269, 2020.

[Lee *et al.*, 2021] Geon Lee, Minyoung Choe, and Kijung Shin. How do hyperedges overlap in real-world hypergraphs? - patterns, measures, and generators. In *WWW*, 2021.

[Leontjeva *et al.*, 2012] Anna Leontjeva, Konstantin Tretyakov, Jaak Vilo, and Taavi Tamkivi. Fraud detection: Methods of analysis for hypergraph data. In *ASONAM*, 2012.

[Park *et al.*, 2009] Youngser Park, C Priebe, D Marchette, and Abdou Youssef. Anomaly detection using scan statistics on time series hypergraphs. In *LACTS*, 2009.

[Ranshous *et al.*, 2017] Stephen Ranshous, Mandar Chaudhary, and Nagiza F Samatova. Efficient outlier detection in hyperedge streams using minhash and locality-sensitive hashing. In *Complex Networks*, 2017.

[Shin *et al.*, 2017] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. Densealert: Incremental dense-subtensor detection in tensor streams. In *KDD*, 2017.

[Shin *et al.*, 2018] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. Patterns and anomalies in k-cores of real-world graphs with applications. *KAIS*, 54(3):677–710, 2018.

[Silva and Willett, 2008] Jorge Silva and Rebecca Willett. Hypergraph-based anomaly detection of high-dimensional co-occurrences. *TPAMI*, 31(3):563–569, 2008.

[Tang *et al.*, 2012] Jie Tang, Bo Wang, Yang Yang, Po Hu, Yanting Zhao, Xinyu Yan, Bo Gao, Minlie Huang, Peng Xu, Weichang Li, et al. Patentminer: topic-driven patent analysis and mining. In *KDD*, 2012.

[Tang *et al.*, 2016] Nan Tang, Qing Chen, and Prasenjit Mitra. Graph stream summarization: From big bang to big crunch. In *SIGMOD*, 2016.

[Yoon *et al.*, 2019] Minji Yoon, Bryan Hooi, Kijung Shin, and Christos Faloutsos. Fast and accurate anomaly detection in dynamic graphs with a two-pronged approach. In *KDD*, 2019.

[Yu *et al.*, 2018] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *KDD*, 2018.

[Zhao *et al.*, 2011] Peixiang Zhao, Charu C Aggarwal, and Min Wang. gsketch: On query estimation in graph streams. *PVLDB*, 5(3):193–204, 2011.

[Zhou *et al.*, 2007] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *NeurIPS*, 2007.