Hyperedge Anomaly Detection with Hypergraph Neural Network

Md. Tanvir Alam
Department of Computer Science and Engineering
University of Dhaka
tanvir15@du.ac.bd

Chowdhury Farhan Ahmed
Department of Computer Science and Engineering
University of Dhaka
farhan@du.ac.bd

Carson K. Leung
Department of Computer Science
University of Manitoba
kleung@cs.umanitoba.ca

Abstract

Hypergraph is a data structure that enables us to model higher-order associations among data entities. Conventional graph-structured data can represent pairwise relationships only, whereas hypergraph enables us to associate any number of entities, which is essential in many real-life applications. Hypergraph learning algorithms have been well-studied for numerous problem settings, such as node classification, link prediction, etc. However, much less research has been conducted on anomaly detection from hypergraphs. Anomaly detection identifies events that deviate from the usual pattern and can be applied to hypergraphs to detect unusual higher-order associations. In this work, we propose an end-to-end hypergraph neural network-based model for identifying anomalous associations in a hypergraph. Our proposed algorithm operates in an unsupervised manner without requiring any labeled data. Extensive experimentation on several real-life datasets demonstrates the effectiveness of our model in detecting anomalous hyperedges.

1 Introduction

Graph-structured data can naturally represent pair-wise relationships, which helps model a wide range of real-life problems. Graph neural network-based machine learning models have been explored extensively for node classification, link prediction, anomaly detection, etc. However, graphs fail to preserve relationships beyond pairs, such as co-authorship networks, social groups, etc. Hypergraphs, on the other hand, can model higher-order complex relationships and associate any number of entities. Neural network-based machine learning models have also been developed for hypergraphs. These models primarily focus on node classification, node clustering, link prediction, etc. Anomaly identification in hypergraphs has received less attention comparatively.

Anomaly detection is a core data mining task that identifies unusual events deviating from the norm. Graph-based anomaly detection has attracted researchers' attention. Methods have been devised to detect anomalies within a single graph [1–3]. Most earlier research uses statistical models or substructure mining, which limits the capability of the methods in terms of scalability and generalization. Recent research has utilized the expressive power of Graph Neural Networks (GNN) for node classification [4], clustering [5], link prediction [6], etc. Consequently, GNNs are applied in graph anomaly detection also [7]. Algorithms have been developed to detect anomalous nodes within a graph [8, 9], with applications such as identifying compromised nodes in a network, detecting spam accounts in social networks, etc. Anomalous edge detection involves spotting unusual connections between nodes [10], crucial for uncovering unusual communication patterns or fraudulent transactions[11]. On the contrary, graph-level anomaly detection identifies anomalous graphs within a set of graphs [12, 13].

Preprint. Preliminary work.

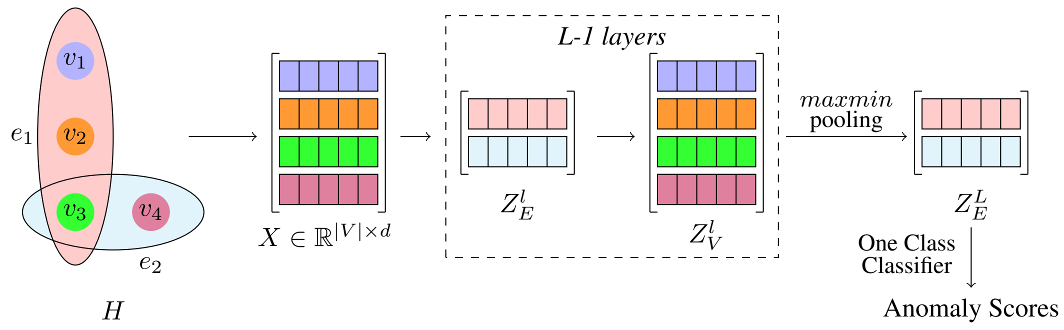

Figure 1: A hypergraph H containing four vertices v_1, v_2, v_3, and v_4. There are two hyperedges e_1 and e_2 where e_1 associates the vertices v_1, v_2, v_3 and e_2 associates the vertices v_3, v_4. The hyperedge embeddings Z_E^l are learned by aggregating the features from the matrix X, and then the node embeddings Z_V^l are derived. At the final level, the hyperedge embeddings are learned by applying $maxmin$ pooling to the node embeddings. Finally, a one-class classifier is applied to find the anomaly scores.

The ability to preserve multi-entity relationships makes hypergraphs a convenient choice for modeling many real-life problems. Research on anomaly identification from hypergraphs aims to detect uncommon higher-order associations represented by hyperedges. Hyperedge anomaly detection has many practical applications, such as identifying abnormal network traffic patterns involving multiple devices, unusual group activities in social networks, fraudulent financial transactions involving numerous parties, and rare interactions among multiple genes. Current research has focused majorly on hashing and statistical similarity measures. LSH[14] devised a similarity-measure-based anomaly detection method for hyperedge streams using minhash and locally sensitive hashing. HashNWalk [15] employs random walk similarities and the hash function to determine anomalies in hyperedge streams. A variational expectation-minimization algorithm has been tailored for hypergraphs to detect anomalies in [16]. None of these approaches integrate node feature information into the process, limiting their generalizability and effectiveness.

Similar to graph neural networks, hypergraph neural networks have proven effective in extracting expressive representations. HGNN [17], HyperGCN [18], AllSet [19] introduced different frameworks for hypergraph neural networks and demonstrated its effectiveness in hypergraph node classification. HCoN[20] developed a model for both node and hyperedge classification. AHP [21] adopts adversarial training to hypergraph neural network for hyperedge prediction. However, the potential of hypergraph neural networks for anomaly detection is still unexplored. To the best of our knowledge, no deep neural network models have been proposed for anomaly detection from hypergraphs in the literature.

Considering the research gaps, in this work, we design a novel hypergraph neural network-based end-to-end model to address the problem of detecting anomalous hyperedges. We propose HAD, a Hyperedge Anomaly Detection algorithm that exploits the attributes or characteristics associated with the hypergraph nodes. HAD is an unsupervised model that operates without the need for labeled data, which is scarce in the domain of anomaly detection. Through the iterative message passing and aggregation processes of the hypergraph neural network, our model effectively propagates information across multiple hops, enabling the utilization of global context and long-range dependencies. We leverage a max-min pooling technique to compute the hyperedge embedding that captures the diversity of the constituent nodes. In Figure 1, we present a flowchart of our proposed method. Our contributions in this work can be summarized as:

- We devise a novel end-to-end hypergraph neural network model to detect anomalous hyperedges.
- Our proposed model operates in an unsupervised manner, without requiring any labeled data for training.
- We curate six real-life hypergraph datasets from different domains for evaluating hyperedge anomaly detection performance.

- We conduct extensive experimentation demonstrating the effectiveness of our method in terms of accuracy in finding anomalous hyperedges.

The rest of the paper is organized as follows: Section 2 discusses the related works, Section 3 defines the problem, and Section 4 presents the proposed model. Section 5 examines the effectiveness of the model, and we conclude the paper in Section 6.

2 Related works

In this section, we discuss the research works related to our work. We focus on hypergraph learning methods and hypergraph anomaly detection.

2.1 Hypergraph Learning Methods

Following the impactful utilization of deep learning to graph-structured data, hypergraph neural network has been devised for learning hypergraph-related tasks. Initially, a spectral hypergraph embedding method based on the hypergraph Laplacian was introduced in [22]. HGNN [17] generalized graph convolutional network [23] to the hypergraph domain by propagating features through a single-stage message-passing framework. HyperGCN [18] also adapted graph convolutional network to hypergraph by approximating the hypergraph with a graph. To properly capture the higher-order relationships in the representations, AllSet [19] proposed a two-stage message-passing framework. In this approach, instead of learning the node representations from the neighborhood, the hyperedge representations are learned from the node representations/features of the previous layer first. Then, the hyperedge representations are aggregated to learn the node representations. HCoN[20] devised a model for both node and hyperedge classification that considers both node and hyperedge from the previous layer. Method for hyperedge prediction [21] based on hypergraph neural networks is explored that adopts generative adversarial training to generate negative examples.

2.2 Hypergraph Anomaly Detection

Anomaly detection methods for graph-structured data can be broadly categorized into four categories: (1) Node Anomaly Detection, (2) Edge Anomaly Detection,(3) Subgraph Anomaly Detection, and (4) Graph-level Anomaly Detection. OCGNN [7] proposes an end-to-end framework for node anomaly detection by extending a one-class support vector machine. BWGNN [8] analyzes spectral energy distributions and devises a graph neural network to detect node anomalies. A scalable approach for detecting anomalies in a dynamic graph setting is proposed in [10]. IGAD [12] introduces a Point Mutual information-based loss function to graph neural network for graph-level anomaly detection. OCGTL [13] develops a one-class graph transformation learning model to detect anomalies from a set of graphs. For hypergraph data, research on anomaly detection has mostly focused on hyperedge level anomalies. An anomaly detection method for hyperedge streams using minhash and locally sensitive hashing was developed by LSH [14]. HashNWalk [15], an incremental algorithm, uses random walk similarities and hash functions to identify anomalies in hyperedge streams. To scale the algorithm for large-scale streams, it maintains a constant-size summary of the stream to calculate the anomaly scores. A variational expectation-maximization algorithm is tailored for hypergraphs to detect anomalies through probability mass function estimation in [16].

3 Preliminaries

In this section, we introduce notations, problem definitions, and preliminaries on hyperedge anomaly detection.

3.1 Notations

A hypergraph can be represented as $H = (V, E, X)$, where $V = \{v_1, v_2, ..., v_{|V|}\}$ is the set of nodes or vertices, $E = \{e_1, e_2, ..., e_{|E|}\}$ is the set of hyperedges, and $X \in \mathbb{R}^{|V|\times d}$ is the feature matrix. Each hyperedge is a subset of the vertices set that it connects, i.e., $\forall_{e\in E} e \subseteq V$. The incidence matrix of hypergraph H is denoted by $A_H \in \mathbb{R}^{|E|\times|V|}$, where the (i, j)-th entry is 1 if the i-th hyperedge contains the j-th vertex, and 0 otherwise. X_v represents the d-dimensional feature vector of the node v.

3.2 Problem Definition

Given a hypergraph $H = (V, E, X)$, the task of hyperedge anomaly detection is to learn a function $f : 2^V \rightarrow [0, 1]$ that assigns an anomaly score to a hyperedge. A higher score indicates a higher likelihood of being an anomaly for a hyperedge. Note that the hyperedge is not necessarily a member of E.

4 Proposed Methods

In this section, we present our proposed model, HAD. HAD learns the node embeddings using a hypergraph neural network. For each hyperedge, we derive its embedding by pooling from the embeddings of the nodes it connects. We define the centroid of the hypergraph as the mean of all the hyperedge embeddings in the hypergraph. Finally, we train a one-class classifier that optimizes the mean Euclidean distance between the hyperedge embeddings and the centroid.

4.1 Learning Node Embeddings

In our model, We begin by learning the node embeddings of the hypergraph using a hypergraph neural network architecture. The vector representation of a hyperedge $e \in E$ at layer l, Z_e^l, is derived from the embeddings of the nodes it contains from the previous layer.

$$Z_e^l = ENN^l(\sum_{v \in e} Z_v^{l-1}) \tag{1}$$

Here, ENN^l is the multi-layer perceptron for hyperedges at layer l. Z_v^{l-1} is the node embedding of the node/vertex v of layer $l-1$. We derive the vector representation of a node $v \in V$ at layer l, denoted as Z_v^l, from the embeddings of the hyperedges containing the node.

$$Z_v^l = VNN^l(\sum_{e \in E_v} Z_e^l) \tag{2}$$

Here, VNN^l is the multi-layer perceptron for nodes at layer l. E_v is the set of hyperedges that contains v. That is, $E_v = \{e : e \in E \text{ and } v \in e\}$. Note that $Z_v^0 = VNN^0(X_v)$, where X_v is the d-dimensional feature vector of the node v from the feature matrix.

4.2 Learning Hyperedge Embeddings

The embedding of a hyperedge e at the final layer L, Z_e^L, is learned by pooling from embeddings of the nodes it connects. We use $maxmin$ pooling, subtracting the element-wise minimum values from the element-wise maximum values, which captures the diversity of the nodes within the hyperedge. This information of diversity within a hyperedge may be crucial for detecting anomalies.

$$Z_e^L = \max\{\bigcup_{v \in e} Z_v^{L-1}\} - \min\{\bigcup_{v \in e} Z_v^{L-1}\} \tag{3}$$

4.3 One Class Classifier

In order to make our model trainable end-to-end, we have developed a one-class classifier by optimizing an objective function. We compute a centroid of the given hypergraph as a reference point for calculating anomaly scores. The centroid, C_H, is calculated by taking the mean of all the hyperedge embeddings in the hypergraph.

$$C_H = \frac{1}{|E|} \sum_{e \in E} (Z_e^L) \tag{4}$$

The anomaly score of a hyperedge e, $f(e)$, is calculated by its Euclidean distance from the embedding Z_e^L to the hypergraph centroid C_H.

$$f(e) = \|Z_e^L - C_H\|_2 \tag{5}$$

Objective Function: Considering the hyperedges in E as inliers, our objective is to minimize their anomaly scores. Our model trains the one-class classifier by minimizing the mean anomaly score over all the hyperedges in the hypergraph. The objective function is defined as follows,

$$\min \frac{1}{|E|} \sum_{e \in E} \|Z_e^L - C_H\|_2 \tag{6}$$

In our approach, rather than fixing the centroid C_H as done in existing one-class classifiers[24], we dynamically update the centroid based on the changing values of Z_e^L for $e \in E$ during training. To prevent the "hypersphere collapse" issue, as discussed in [24], we introduce a hyperparameter called $loss\_threshold$. When the loss value falls below $loss\_threshold$, we stop further optimization to prevent hypersphere collapse.

We present the pseudocode of the training phase of HAD in Algorithm 1. In line 1, we calculate the initial node embeddings. Then, we learn the layer-wise hyperedge and node embeddings in lines 5 and 6, respectively. In line 9, we calculate the hyperedge embeddings of the final layer by applying the $maxmin$ function. After computing the centroid, C_H, and $loss$ in lines 11 and 12, we update the parameters of the multi-layer perceptron in line 13.

Algorithm 1: HAD Training

```
Input  : $H = (V, E, X)$: A hypergraph, $loss\_threshold$: The loss threshold
Output : $Z_V^{L-1}$: The node embeddings, $C_H$: The centroid of the hypergraph $H$
1  begin
2      while True do
3          $Z_V^0 \leftarrow VNN^0(X)$;
4          for $l \leftarrow 1 \rightarrow L-1$ do
5              $Z_E^l \leftarrow ENN^l(A_H Z_V^{l-1})$;
6              $Z_V^l \leftarrow VNN^l(A_H^T Z_E^l)$;
7          end
8          for $e \in E$ do
9              $Z_e^L \leftarrow \max\{\bigcup_{v \in e} Z_v^{L-1}\} - \min\{\bigcup_{v \in e} Z_v^{L-1}\}$;
10         end
11         $C_H \leftarrow \frac{1}{|E|} \sum_{e \in E} (Z_e^L)$;
12         $loss \leftarrow \frac{1}{|E|} \sum_{e \in E} \|Z_e^L - C_H\|_2$;
13         if $loss \leq loss\_threshold$ then
14             Break;
15         Update the parameters of $\bigcup_{i=0}^{L-1} VNN_i$ and $\bigcup_{i=1}^{L} ENN_i$ to minimize $loss$;
16     end
17 end
```

In algorithm 2, we present the pseudocode of the function to calculate the anomaly score of a given hyperedge. It computes the hyperedge embedding by applying the $maxmin$ function to the final embeddings of the nodes that the hyperedge contains and then returns the distance from the centroid as the anomaly score.

Algorithm 2: Anomaly Score Prediction

```
Input  : $e \subseteq V$: A hyperedge, $H = (V, E, X)$: A hypergraph, $C_H$: The centroid of the
         hypergraph $H$, $Z_V^{L-1}$: The node embeddings
Output : $score$: the anomaly score of hyperedge $e$
1  begin
2      $Z_e^L \leftarrow \max\{\bigcup_{v \in e} Z_v^{L-1}\} - \min\{\bigcup_{v \in e} Z_v^{L-1}\}$;
3      $score = \|Z_e^L - C_H\|_2$;
4  end
```

5 Experimental Results

In this section, we present the experimental settings and perform an extensive result analysis of our proposed algorithm. Section 5.1 describes the experimental setup. Section 5.2 presents the details of the datasets. Section 5.3 discusses the baseline considered for comparison and Section 5.4 analyzes the experimental results. Finally, in Section 5.5, we visualize the anomaly scores for inlier and anomalous hyperedges.

5.1 Experimental Setup

To evaluate the performance of our algorithm, we conducted experiments on six real-world datasets. We implemented the algorithm using Python3 programming language and utilized an Intel Core i7-6700k CPU @ 4.00 GHz with 16 GB RAM to conduct the experiments. We split the inlier hyperedges for each dataset into a training set (80%) and an inlier test set (20%). The training set containing inlier hyperedges is only used to train the model. In contrast, the test set contains anomalous hyperedges and an oversampled inlier test set to address the class imbalance issue. We measured the AUROC value for performance metrics and used five-fold cross-validation, considering the mean value over five runs. We set the number of layers L to two and the $loss\_threshold$ to 0.0001 for all the datasets. Note that we only require labeled data to test the performance of our algorithm, and we avoid using any validation set since the model we proposed is unsupervised.

5.2 Datasets

We have collected six real-life hypergraph datasets for our experiments from different domains. In Table 1, we present the statistical summary of the datasets, and the dataset descriptions are provided below.

Mushroom: It contains information about various mushroom species. For each species, 22 nominal value attributes are recorded. The species are categorized as either edible or poisonous. We created a hypergraph using the nominal values as nodes, with each hyperedge representing a species connecting the nodes of its nominal values. The edible species are considered as inliers, whereas the poisonous species are considered as anomalies. Due to the absence of any node features, we assigned each node with unique identity vectors as features.

Co-citation datasets: Citeseer, Cora, and Pubmed are three co-citation datasets containing information about papers, their citations, and co-citations. In the hypergraph representation, Each node represents a paper, and each hyperedge connects papers cited in another paper. Bag-of-words features from the paper abstracts are used as node features. For hyperedge classification, on datasets where labels are unavailable, the labels of the nodes in a hyperedge are used to label the hyperedge [20]. Following this approach, we also utilized the node labels. We considered the hyperedges containing a node labelled with the most frequent node label as inliers and all the other hyperedges as anomalies.

Authorship datasets: CoraA and DBLP are two authorship datasets we considered in our experiments. Each node represents a paper in these hypergraphs, and the hyperedges connect papers authored by a specific author. Similar to co-citation datasets, bag-of-words features of the abstracts are used as node features. The same strategy as co-citation datasets is used to distinguish the anomalous hyperedges.

Table 1: Statistics of the real-life datasets.

Dataset	Mushroom	Citeseer	CoraA	Cora	Pubmed	DBLP
Number of nodes, $\|V\|$	117	1,458	2,388	1,434	3,840	41,302
Number of hyperedges, $\|E\|$	8,124	1,079	1,072	1,579	7,963	22,363
Number of train hyperedges	3,133	242	350	272	3,604	2,897
Number of test hyperedges	8,416	1,554	1,270	2,480	6,916	37,484
Average hyperedge size	22.00	3.20	4.27	3.03	4.34	4.45
Maximum hyperedge size	23	27	44	6	172	203

5.3 Baselines

Our experiments considered three baseline methods for performing comparative performance analysis. We applied LSH [14], an algorithm proposed for anomaly detection in a hyperedge stream, by randomly shuffling the order of hyperedges. Similarly, we have considered HashNWalk [15], another anomaly detection algorithm for hyperedge streams. We implemented VEM [16], a variational expectation-minimization algorithm, to detect hyperedge anomalies. To calculate the AUROC score, we first normalized the anomaly score for all the algorithms. Furthermore, we have implemented two variations of our algorithm. First, we utilized mean pooling instead of $maxmin$ pooling (Equation 3) in the final layer to examine the significance of capturing diversity in detecting anomalies and named the model HAD-Mean. Second, we fixed the centroid C_H as proposed in existing one-class classifiers[24] and named the model HAD-Fixed. However, for HAD-Fixed, the $loss$ value (Algorithm 1, line 13) converges below the $loss\_threshold$ value of 0.0001 as used in our experimental setup. We have run the training for 1000 epochs to deal with the issue instead of using the $loss\_threshold$ value.

5.4 Results

In Table 2, we present the AUROC scores in percentages of our model along with the baselines for all six datasets. The best result of all models is highlighted in bold. We observe that our model has outperformed all the baselines significantly on all the datasets. On dataset Mushroom, HAD-Mean and HAD (Proposed) both achieve perfect scores of 100%, significantly outperforming other methods. VEM is the next best, with a score of 72.04%. The AUROC score for HashNWalk is 57.47%, while LSH performs poorly at 32.02%. The AUROC score of our model is 27.95% more than that of VEM, the highest among the baselines. On Citeseer, the AUROC score is 8.49% more than LSH, the best-performing baseline. The improvements are also significant for CoraA and Cora, which are 13.94% and 10.92%, respectively. On Pubmed, the difference is marginal compared to HashNWalk (0.03%). For the hypergraph DBLP, the improvement is 2.92%. The better performance is also evident in the comparison with the variants HAD-Mean and HAD-Fixed. HAD-Fixed performs the worst among the HAD variants, particularly on datasets Mushroom, Citeseer, CoraA, and DBLP. HAD-Mean performs well on certain datasets, particularly Mushroom and Cora, but fails to match the overall performance of HAD-Proposed. The better AUROC score of our proposed model than HAD-Mean demonstrates the importance of the knowledge of diversity within a hyperedge for anomaly detection. The AUROC score of HAD-Fixed is also lower than our proposed model. Compared to HAD-Mean, the AUROC score is lower on all the datasets except Pubmed for HAD-Fixed. When the centroid is fixed, it becomes challenging for the model to converge and learn due to the possibility of selecting a poor centroid. Having a dynamic centroid allows the model to learn more easily by choosing a suitable centroid, leading to improved performance.

Table 2: AUROC score(%) of our method and baseline methods on real-life datasets.

Dataset	Mushroom	Citeseer	CoraA	Cora	Pubmed	DBLP
LSH	32.02	63.09	46.25	50.40	53.35	48.45
HashNWalk	57.47	48.63	50.07	52.57	59.04	53.65
VEM	72.04	52.20	50.76	55.39	54.07	50.18
HAD-Mean	**100.00**	54.24	39.83	65.72	36.37	46.08
HAD-Fixed	50.01	29.53	29.71	33.12	57.07	16.51
HAD (Proposed)	**100.00**	**71.56**	**64.70**	**66.31**	**59.07**	**56.58**

In Figure 2, we present the loss values over the first 1,000 epochs for all six datasets. We compare two methods, HAD-Fixed and our proposed HAD algorithm, to demonstrate the effect of updating the centroid dynamically. For both methods, we started the training with the same set of parameters initialized randomly and recorded the loss values over epochs. For all the datasets, the loss value decreases sharply in the initial epochs and gradually stabilizes. The decrement in loss value is comparatively larger for HAD (Proposed) than HAD-Fixed, and so the model converges fast and results in better performance. After the initial epochs, although having a higher loss value, the HAD-Fixed model fails to minimize the loss value significantly. For dataset Mushroom (Figure 2-(a), the loss value starts at 74.63 for both methods, as the initial parameters of the models are the same.

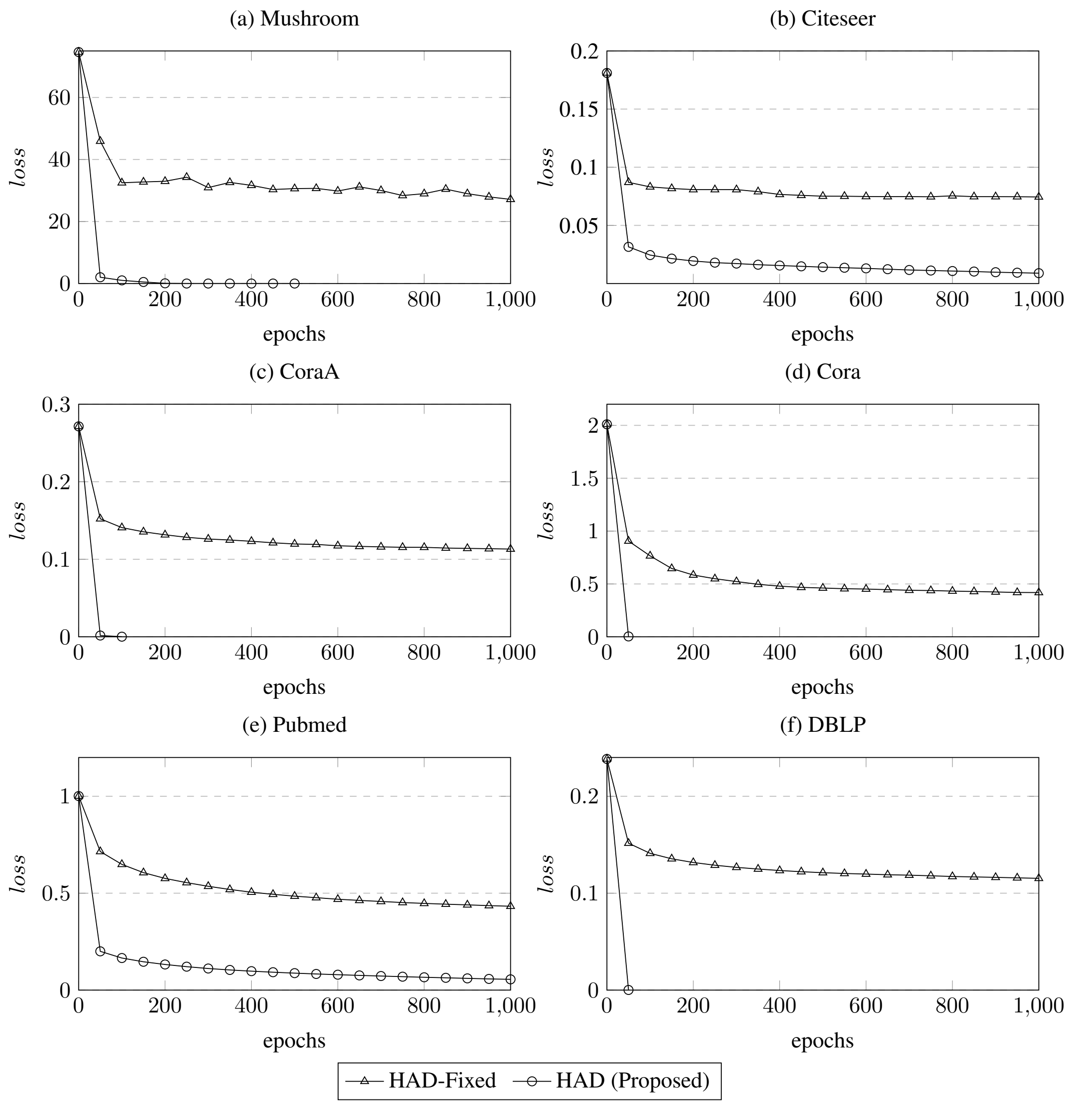

Figure 2: Loss value analysis over epochs

Then, after 50 epochs, for HAD-Fixed, the loss value drops to 45.90, whereas for HAD (Proposed), it decreases to 2.02. The comparatively higher drop in loss is the result of dynamically updating the centroid. After around 500 epochs, for the HAD (Proposed) algorithm, the loss value goes below the $loss_threshold$, and the algorithm terminates. On the contrary, the loss value is 27.13 after 1,000 epochs for HAD-Fixed. This demonstrates that using a fixed centroid makes it hard for the model to converge. A similar trend is also evident for the other datasets. For dataset Citeseer (Figure 2-b), after 1,000 epochs, the loss value decreases to 0.074 for HAD-Fixed, whereas for HAD (proposed), the loss value is 0.008 only. For CoraA, the HAD (proposed) algorithm converges after around 100 epochs, but the loss value for HAD-Fixed is 0.113 after 1,000 epochs, which is relatively higher. For dataset Cora (Figure 2-d), HAD (proposed) converges after around 50 epochs, but the loss value is still higher after a thousand epochs. For dataset Pubmed (Figure 2-e), the loss values are 0.432 and 0.055 for HAD-Fixed and HAD (proposed). For dataset DBLP (Figure 2-f), HAD (proposed) converges after around 50 epochs, but the loss value is 0.115 after a thousand epochs.

5.5 Visualization

In Figure 3, we visualize the anomaly scores of the hyperedges for all six datasets. In most cases, the anomaly scores for inliers are clustered close to zero, while anomalies tend to have higher scores.

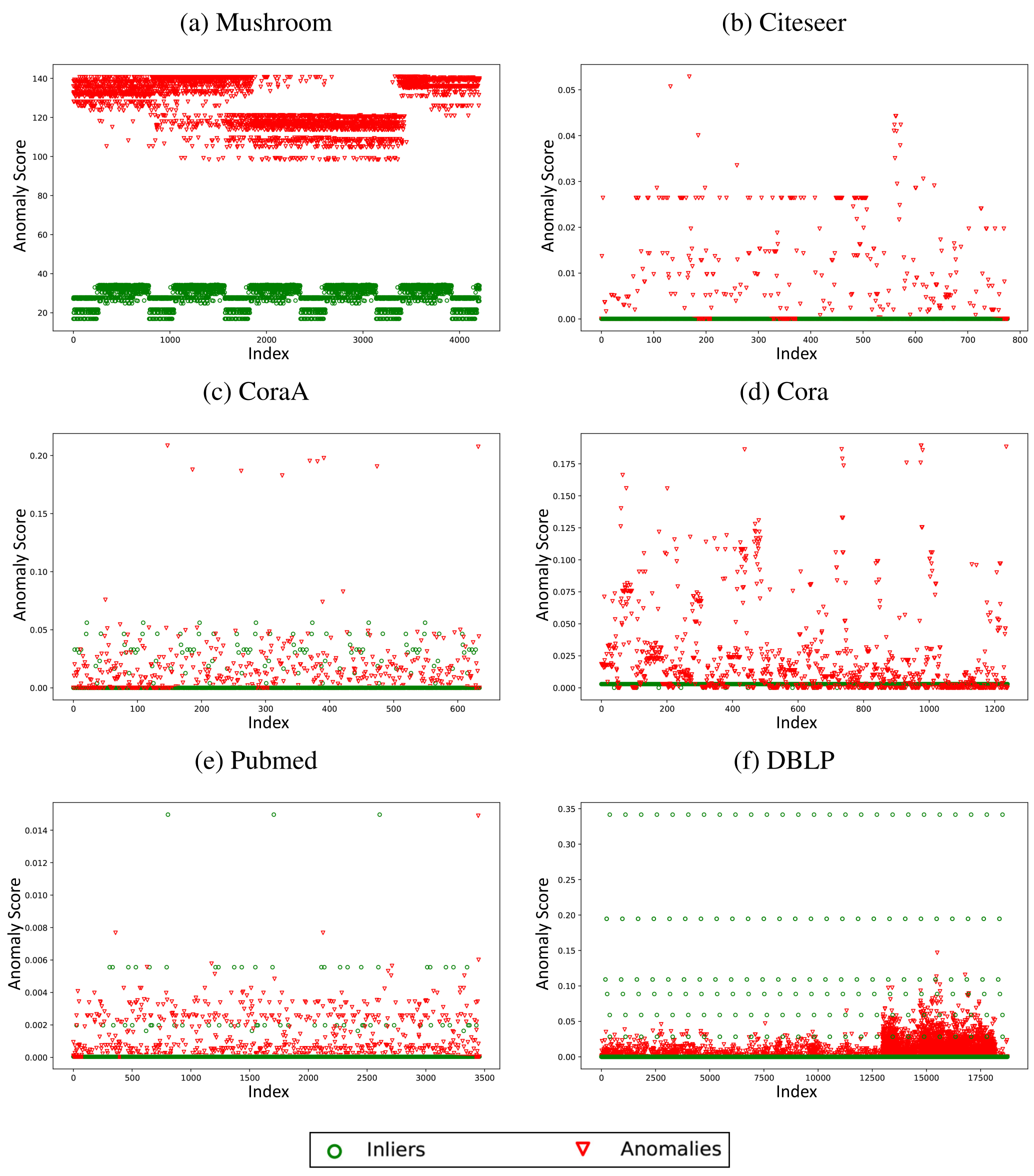

Figure 3: Visualization of Anomaly Scores

In 3-a, for the dataset Mushroom, a clear separation is evident between the anomaly scores for the anomalies and inliers. The clear separation of anomaly scores reflects the higher AUROC score of 100% for this dataset. For dataset Citeseer (Figure 3-b), the anomaly scores are relatively spread out, but anomalies are still distinguishable from inliers. For datasets CoraA and Cora (Figure 3-c and Figure 3-d), inliers are clustered near the bottom, but anomalies are scattered over a wider range of scores. In the case of Pubmed (Figure 3-e), anomalies have higher scores, while inliers are more uniformly distributed near zero. In dataset DBLP (Figure 3-f), many anomalous hyperedges have higher scores than inliers. However, there is some overlap near the bottom, and some of the inliers have higher anomaly scores than anomalies. This justifies the relatively lower AUROC score than other datasets.

6 Conclusion

In this paper, we proposed HAD, an anomaly detection algorithm for hyperedges in a hypergraph. Our proposed model is unsupervised and needs no labeled data. We devised an end-to-end model that employs a hypergraph neural network to learn hyperedge representations and then predicts the anomaly score with a one-class classifier. To the best of our knowledge, we are the first to propose a deep neural network-based model for anomaly detection on hypergraphs. We collected six real-life hypergraph datasets from different domains to evaluate the performance. We performed a comparative result analysis of our method against the state-of-the-art research. A significantly higher AUROC score across the datasets demonstrates the effectiveness of our algorithm. Future research directions for this work may include utilizing labeled data, entire anomalous hypergraph detection, etc. The source code of our algorithm implementation is available online[1]

References

[1] Caleb C Noble and Diane J Cook. Graph-based anomaly detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636, 2003. 1

[2] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Advances in Knowledge Discovery and Data Mining, PAKDD*, pages 410–421. Springer, 2010.

[3] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. Spotlight: Detecting anomalies in streaming graphs. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1378–1386, 2018. 1

[4] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016. 1

[5] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. Graph clustering with graph neural networks. *Journal of Machine Learning Research*, 24(127):1–21, 2023. 1

[6] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018. 1

[7] Xuhong Wang, Ying Du, Ping Cui, and Yupu Yang. Ocgnn: one-class classification with graph neural networks. *CoRR*, abs/2002.09594, 2020. 1, 3

[8] Jianheng Tang, Jiajin Li, Ziqi Gao, and Jia Li. Rethinking graph neural networks for anomaly detection. In *International Conference on Machine Learning*, pages 21076–21089. PMLR, 2022. 1, 3

[9] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S Yu. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pages 315–324, 2020. 1

[10] Stephen Ranshous, Steve Harenberg, Kshitij Sharma, and Nagiza F Samatova. A scalable approach for outlier detection in edge streams using sketch-based approximations. In *Proceedings of the 2016 SIAM international conference on data mining*, pages 189–197. SIAM, 2016. 1, 3

[11] Ge Zhang, Zhao Li, Jiaming Huang, Jia Wu, Chuan Zhou, Jian Yang, and Jianliang Gao. efraudcom: An e-commerce fraud detection system via competitive graph neural networks. *ACM Transactions on Information Systems (TOIS)*, 40(3):1–29, 2022. 1

[12] Ge Zhang, Zhenyu Yang, Jia Wu, Jian Yang, Shan Xue, Hao Peng, Jianlin Su, Chuan Zhou, Quan Z Sheng, Leman Akoglu, et al. Dual-discriminative graph neural network for imbalanced graph-level anomaly detection. *Advances in Neural Information Processing Systems*, 35:24144–24157, 2022. 1, 3

[13] Chen Qiu, Marius Kloft, Stephan Mandt, and Maja Rudolph. Raising the bar in graph-level anomaly detection. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 2196–2203. International Joint Conferences on Artificial Intelligence Organization, 2022. 1, 3

[1]https://github.com/tfahim15/HAD2024

[14] 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: Proceedings of Complex Networks 2017 (The Sixth International Conference on Complex Networks and Their Applications)*, pages 105–116. Springer, 2018. 2, 3, 7

[15] 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-22*, pages 2129–2137. International Joint Conferences on Artificial Intelligence Organization, 7 2022. 2, 3, 7

[16] Jorge Silva and Rebecca Willett. Hypergraph-based anomaly detection of high-dimensional co-occurrences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(3):563–569, 2008. 2, 3, 7

[17] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3558–3565, 2019. 2, 3

[18] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. Hypergcn: A new method for training graph convolutional networks on hypergraphs. *Advances in neural information processing systems*, 32, 2019. 2, 3

[19] Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. You are allset: A multiset function framework for hypergraph neural networks. In *International Conference on Learning Representations*, 2022. 2, 3

[20] Hanrui Wu, Yuguang Yan, and Michael Kwok-Po Ng. Hypergraph collaborative network on vertices and hyperedges. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45 (3):3245–3258, 2023. 2, 3, 6

[21] Hyunjin Hwang, Seungwoo Lee, Chanyoung Park, and Kijung Shin. Ahp: Learning to negative sample for hyperedge prediction. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2237–2242, 2022. 2, 3

[22] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. *Advances in neural information processing systems*, 19, 2006. 3

[23] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 3

[24] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4393–4402, 10–15 Jul 2018. 5, 7