Self-supervised anomaly detection in static attributed graphs

Benoît Corsini, Pierre-André Noël, David Vázquez, Perouz Taslakian

Abstract

In this work, we consider the task of detecting anomalous nodes in static attributed graphs in the absence of labeled samples. Anomaly (or outlier) detection is generally a difficult problem to solve. This is due to the absence of a concrete definition of what constitutes an anomaly, but also because anomalies themselves occur very rarely and finding enough samples to learn from is often impossible. In this work, we propose two methods that combine Graph Neural Networks (GNNs) with a self-supervised training scheme to learn a representation of normal (non-anomalous) nodes in the graph, and identify anomalies using these learned representations. Both models, called *HideGNN* and *MaskGNN*, use a mask-based self-supervision approach, whereby the algorithm is set up to predict the masked or hidden features of nodes at train time. We show experimentally that our approach outperforms known baselines on real-world datasets and we explore the power of the methods in detecting specific types of anomalies with synthetic data.

1 Introduction

Anomaly detection refers to the problem of identifying the samples in a dataset that are "out of the ordinary", or finding an observation that "differs so much from other observations as to arouse suspicion that it was generated by a different mechanism" [16]. The problem itself is ill-posed, as the definition of what constitutes an anomaly (or an *outlier*) is vague and often depends heavily on the application domain. As such, anomaly detection is a challenging task to solve, which is made even more daunting by the fact that anomalies are extremely rare in practice and labeled data samples are effectively non-existent. Nevertheless, anomaly detection is an important task that has been studied extensively and has applications in many areas such as healthcare [21], finance [39], cybersecurity [15, 30], and sensor networks [8].

In this work, we are interested in detecting anomalies in static graph-structured datasets where labeled anomalous samples are not available at train time. More precisely, we want to identify the *anomalous nodes* of the input graph; these are the nodes that differ significantly from the majority of remaining nodes in the graph, in terms of their features, their relations to other nodes, or both. The ability to accurately find anomalies enhances our capacity to solve many common real-life problems, such as identifying bots or spammers in a social network, detecting fraud in financial transactions, identifying outliers in groups of people, etc. Moreover, having a good anomaly detection algorithm on graphs can be useful in improving other graph-based problems, such as clustering or classification.

The challenge of anomaly detection in unstructured data (such as a collection of images) differs slightly from that of identifying outliers in graph datasets, where an explicit relationship is defined between sample pairs. While in the former case each data object can be treated independently, nodes in a graph-structured dataset exhibit explicit inter-dependencies that need to be considered during the detection process. The nature of anomalies in a graph can, in addition, be relational: a data object that may be anomalous due solely to its own features, may be highly unusual if we consider its relationships with other nodes (e.g., its neighbors in the graph). Thus, in a graph with node features, anomalies come in different forms (related — with important differences[1]— to the three cases of Bojchevski and Günnemann [2]).

- **Feature-based anomalies:** anomalous nodes can be identified when considering only the node features, disregarding the structure of the graph. For example, some nodes may have unusually large features, or unexpected combinations of features.
- **Graph-based anomalies:** anomalous nodes can be identified when considering only the graph structure. For example, some nodes may have unusually low or high degree, or may have neighbors belonging to unusual combinations of clusters.
- **Joint anomalies:** anomalous nodes can only be identified when *jointly* considering node features and graph structure. For example, a node could have commonly-seen features and belong to a typical graph community, yet be anomalous due to those features being unusual for that specific graph community.

[1]The three cases discussed in [2] are not mutually exclusive. Cases 1 and 3 both imply feature-based anomalies; cases 2 and 3 both imply graph-based anomalies. Our joint anomalies are not explicitly addressed in [2], but the PAICAN model it introduces can detect them to some extent (see Figures 4 and 5).

As the graph structure is an important consideration when detecting anomalies, techniques that perform well in finding anomalies in unstructured data may perform poorly when the dataset is relational. Therefore, to have a better handle on the problem, methods that take into account the neighborhood information of nodes are needed.

One common approach to anomaly detection is learning the distribution of the input dataset [25]. This approach hinges on the assumption that anomalies are extremely rare, and most of the data objects are in fact "normal" samples. Hence, by learning the distribution of the input samples, one can expect to approximate the overall distribution of the dataset reasonably well. The likelihood of each sample can then be obtained, and the ones with lowest likelihood are classified as anomalous. We follow this approach for detecting anomalies in graph-structured data, with the distinction in the way we approximate the distribution of the input graph: by taking into account the explicit relationships between nodes.

Learning from graph-structured data has garnered a lot of attention recently [27, 8, 9], and research on the topic has intensified due to recent abundance of datasets in applications such as social networks, cybersecurity, and online retail. Subsequently, the development of neural architectures, such as *Graph Neural Networks* (GNNs) [7, 22, 10, 14], has shown promising results in tasks like node classification and link prediction, fanning the research flames even further. GNNs learn a representation of the graph through a message-passing scheme [10], whereby the representation of a node is updated at every step of the algorithm based on the representation of its neighbors. These learned representations are invariant to the ordering of the nodes within the dataset, as well as around the neighborhoods of each node.

It is in this context (further detailed in Section 2) that we propose two self-supervised learning models for anomaly detection: HideGNN and MaskGNN (Section 3). We conduct experiments on real-world datasets and discuss them in detail in Section 4. We also generate a synthetic dataset and use it to explore the capacity of our methods for detecting joint anomalies; we present these results in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

Detecting anomalies in structured data is a topic studied in different disciplines, and various techniques have been developed to tackle the problem. In this work, we concentrate on anomaly detection in static graph-structured data where labels are not available (unsupervised). We group related work into the following three categories.

2.1 Anomaly detection on graphs.

Anomaly detection in graphs has been a focus of researchers for the last two decades [1]. Problems using anomaly detection methods on graphs include identifying spam [3, 35, 26], finding click fraud [19, 23], and detecting malware [18, 28, 37]. Recent methods in this area have addressed the problem by comparing the topological information of each node with their features, and possibly with those of their neighbors. Two particular methods have shown promising results: (a) analyzing graphs at multiple scales and identifying anomalies at each of these scale [13], and (b) combining a clustering algorithm with an anomaly detection algorithm to highlight several types of anomalies [2]. These two methods assume that anomalous nodes strongly depend on the interactions between the graph topology and the node features.

2.2 Anomaly detection with neural networks.

To our knowledge, there is little body of work in the deep neural network literature that focuses on anomaly detection in graphs. However, there exist multiple algorithms on anomaly detection on non-structured data (e.g., in images or time-series [4]), that often use energy-based methods [25, 47]. More recently, a combination of energy-based methods and generative adversarial networks [11, 12, 5] has shown promising results in finding anomalies [24].

2.3 Self-supervised learning on graphs.

Graph neural networks were first introduced by Kipf and Welling [22] and research in this area has seen great progress since. These methods are very successful in learning effective representations of relational data. GNN architectures are designed to capture the neighborhood information of every node and generate embeddings that are sensitive to local graph structure. These embeddings are used to efficiently solve several tasks such as node classification [41, 14, 38], graph classification [44, 32], and link prediction [43, 40, 6]. Over the last few years, methods for training GNNs without supervision have attracted more attention [27, 42], especially due to the high cost of labelling datasets. Many such methods usually learn relevant representations of the input graph as a pre-training step, then apply the downstream task (such as node classification) methods. When properly used, they show state-of-the-art results on the three tasks mentioned above: node classification [20, 29], graph classification [46], and link prediction [36].

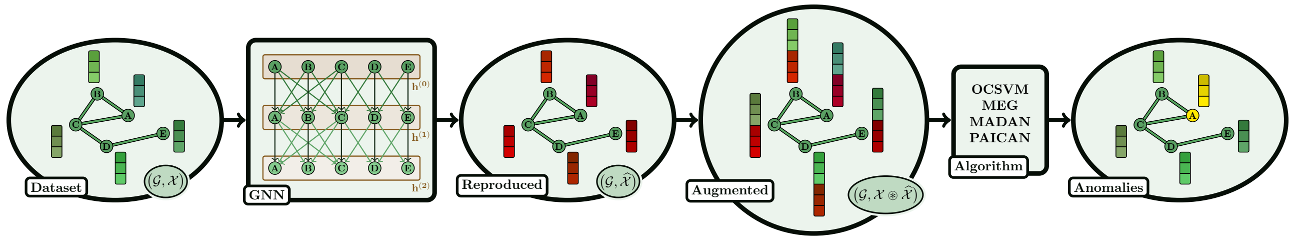
Dataset
$(\mathcal{G}, \mathcal{X})$
GNN
$h^{(0)}$
$h^{(1)}$
$h^{(2)}$
Reproduced
$(\mathcal{G}, \widehat{\mathcal{X}})$
Augmented
$(\mathcal{G}, \mathcal{X} \circledast \widehat{\mathcal{X}})$
OCSVM
MEG
MADAN
PAICAN
Algorithm
Anomalies

Figure 1: A representation of the method of this paper. We first use the power of Graph Neural Networks to create a reproduction (in red) of the node features (in green). This reproduction is based on the features of the neighborhoods of the nodes and uses the general distribution of the features to reach its optimum. These reproduced features are then used to augment the original dataset, which is then put as input of standard anomaly detection algorithms. These method show state-of-the-art results and greatly improves the performances of the algorithms (see Table 2).

3 Proposed Method

In this section, we present our general framework for detecting anomalies in graph-structured datasets, also summarized in Figure 1. This framework contains a self-supervised learning step, for which we describe two methods that use GNNs to learn node representations. With this framework, we first train a model in a self-supervised manner with the input dataset $(\mathcal{G}, \mathcal{X})$ (consisting of a graph structure $\mathcal{G}$ and a set of node features $\mathcal{X}$) in order to reproduce the node features. The reproduced features $\widehat{\mathcal{X}}$ are an approximation of the original features $\mathcal{X}$ and are combined with $\mathcal{X}$ using some operator $\circledast$ to obtain an *augmented* set of features. We finally apply an existing anomaly detection algorithm to the augmented dataset $(\mathcal{G}, \mathcal{X} \circledast \widehat{\mathcal{X}})$. Note that the original and augmented datasets have the same graph structure, but differ in their node features.

Conceptually, the proposed method first learns what a "normal" node in the graph looks like, using both the node features and the local neighborhood of each node, then classifies the nodes that are far from that normal as anomalous. This approach accommodates the two important assumptions that (1) anomalies are extremely rare, and thus the input graph consists almost entirely of normal nodes; and (2) we are not given a definition of anomalous nodes, nor any examples of such nodes in the input dataset. In this work, we consider undirected and unweighted graphs, but all our algorithms can also be applied (with minimal adjustment) to directed graphs with weighted edges.

In what follows, we present two GNN-based models that perform feature-estimation task described above through self-supervision on the node features of the input graph. Both methods rely on masking out some of the features of the input nodes at train time, rewarding the model when masked features are correctly predicted. The two methods differ by the choice of how this feature masking occurs during the learning iteration. Once the model has learned to approximate the input graph, we use this learned function to compute a measure of how anomalous each node is (detailed in Section 4).

3.1 Graph Neural Networks

Let $\mathcal{G} = (V, E)$ be a graph with a set V of nodes and a set E of undirected edges connecting pairs of nodes in V, along with a set of node features $\mathcal{X} = \{X_v : v \in V\}$ where $X_v \in \mathbb{R}^{d_f}$ is the d_f-dimensional feature of node $v \in V$. We write $X_v[j]$ for the j-th entry of the vector X_v.

Graph Neural Networks (GNNs) are multi-layered neural networks N that use the graph and node feature information to generate embeddings $h_v \in \mathbb{R}^{d_h}$, for all $v \in V$, using a message-passing scheme that aggregates neighboring representations at each layer. We let $h_v^{(\ell)} \in \mathbb{R}^{d_h^{(\ell)}}$ be the hidden representation of node $v \in V$ learned at the ℓ^{th} layer, and initialize $h_v^{(0)} = X_v$ for all $v \in V$. The *neighborhood* of a node v is the set of all its adjacent nodes, that is $\mathcal{N}(v) = \{u \mid (u, v) \in E\}$.

A GNN with L layers updates the hidden features $h_v^{(\ell)}$ at layer ℓ for all the vertices $v \in V$ simultaneously (for $\ell = 0, \ldots, L-1$).

$$h_v^{(\ell+1)} = \textsc{Update}^{(\ell)}\Big(\textsc{Aggregate}^{(\ell)}\Big(\big\{h_u^{(\ell)} : u \in \mathcal{N}(v)\big\}\Big)\Big).$$

The function Aggregate combines the representations of v and its neighbors obtained from the previous layer, and Update uses this combined embedding to update the hidden representation of node v. Aggregate is most commonly defined as the sum of the neighbor nodes multiplied by a matrix of weights, but other possible functions exist, such as weighted sum [22], set pooling [45], or neighborhood attention [41]. Update is generally a non-linear function.

The two models we describe in subsequent sections use GNNs with $L = 2$ layers. We define the Aggregate function as the sum of the hidden representations of the neighboring nodes multiplied by a matrix of weights. Our Update function is a simple ReLU activation function for the first layer, and the identity for the second layer. Choosing the identity for the second layer allows us to have negative values as the output of our models, thus ensuring the capacity to reproduce the original features.

In summary, we use GNNs N that take the hidden representation $h^{(0)} = \mathcal{X}$ as input and output $h^{(2)} = N(\mathcal{X})$, with

$$h_v^{(2)} = W_1 \cdot h_v^{(1)} + C \cdot W_2 \cdot \sum_{u \in \mathcal{N}(v)} h_u^{(1)}$$

and

$$h_v^{(1)} = \sigma \left(W_3 \cdot h_v^{(0)} + C \cdot W_4 \cdot \sum_{u \in \mathcal{N}(v)} h_u^{(0)} \right) ,$$

where $\sigma(x) = \max(0, x)$, $C \in \{0, 1\}$ is a hyperparameter, and (W_1, W_2, W_3, W_4) constitute the learned parameters of the model.

In what follows, we introduce two different GNN algorithms for reproducing the node features $\mathcal{X}$. We denote such reproduced features as $\widehat{\mathcal{X}}$.

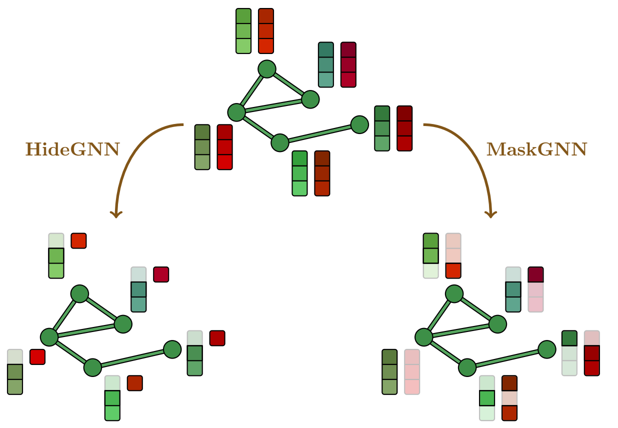

Figure 2: How HideGNN and MaskGNN differ in the way they train. The node features (green blocks) are used by the GNN to learn an embedding for each node (red blocks). HideGNN (left) hides one feature (the top feature in this example) and learns a reproduction of the hidden feature (in red). By hiding one feature after the other, we are able to reproduce the original features. MaskGNN (right) masks random features over the whole dataset (the shaded features in this example), before only optimizing the embeddings where there was a mask (the non-shaded embeddings in red). By randomly changing the mask, we optimize the reproduced features over all possible values, while avoiding trivial solutions.

3.2 HideGNN

Our first model, called *HideGNN*, uses multiple GNNs to predict the node features, with each GNN specializing in a specific feature. Concretely, each GNN receives as input the original graph together with a version of the input features in which one of the features is *hidden*; the goal of the GNN is to predict the hidden features (Figure 2, left).

Let $X_u^j \in \mathbb{R}^{n_f - 1}$ be the features of node u such that the j^{th} feature is removed, and let $\mathcal{X}^j = \{X_u^j \mid u \in V\}$. Define $N^1, \ldots, N^{n_f}$ to be a set of GNNs such that N^j takes as input $\mathcal{X}^j$ (together with the graph $\mathcal{G}$) and outputs its prediction for the j^{th} (hidden) features of the nodes in V. The loss function for each GNN is defined as the L^2 norm (squared distance) between the predicted and actual features

$$\Phi\big(N^j(\mathcal{X}^j)\big) = \sum_{v \in V} \Big(N^j(\mathcal{X}^j)_v - X_v[j] \Big)^2 .$$

The final prediction $\widehat{\mathcal{X}}$ of every node features is set as the concatenation of the features given by the different GNNs $N(\mathcal{X})[j] = N^j(\mathcal{X}^j)$, and the model loss function is defined as the following sum

$$\mathcal{L}\big(N(\mathcal{X})\big) = \sum_{j=1}^{n_f} \Phi\big(N^j(\mathcal{X}^j)\big) . \qquad (1)$$

The HideGNN model is powerful in representing node features but it is slow to train, since it uses multiple GNNs.

3.3 MaskGNN

Our second model, called *MaskGNN*, uses a single GNN N that takes as input the graph $\mathcal{G}$ and all the features $\mathcal{X}$ and directly outputs a reproduction of the features. During the training phase, it *masks* some of the input features for each node sample and tries to predict the masked value (Figure 2, right).

Define the mask M_v of node $v \in V$ to be a binary vector of length n_f, and let $\mathcal{X}^M$ be the set of *masked features* such that the j^{th} feature of X_v^M is masked if and only if the bit at position j of the mask is zero, that is $M_v[j] = 0$. At train time, the model generates a random mask for each node feature, and learns to predict the masked values. The loss function of this model is as follows.

$$\mathcal{L}\big(N(\mathcal{X})\big) = \sum_{v \in V} \Big(M_v \cdot \big(N(\mathcal{X}^M)_v - X_v\big) \Big)^2 . \qquad (2)$$

By going over multiple random masks M, this algorithm learns a general (non-trivial) representation of the features. The final prediction $\widehat{\mathcal{X}}$ of the node features is the output of N when applied to the unmasked $\mathcal{X}$ (or, equivalently, when the mask M is all true).

The MaskGNN model is fast to train and uses a single GNN; however the model has more parameters than HideGNN. As there is no clear advantage of one model over the other when it comes to training efficiency, we consider both models in this paper.

4 Experiments

Our experiments focus on unsupervised anomaly detection, where we evaluate the effect of pre-training the input graph dataset with a GNN approach. We select four anomaly detection methods, and compare their results with and without pre-training using each of HideGNN and MaskGNN.

Datasets. To test our anomaly detection approach, we use three real-life datasets: Books [34], Disney [34], and Enron [31]. Both Disney and Books are co-purchase networks extracted from Amazon Marketplace. Nodes in these datasets have attributes describing properties of online items (e.g., rating, selling price, etc.), combined with ground truth anomalies labeled through two distinct processes. Anomalies in Disney are labeled manually by high school students, where a node (with 32 features) is labeled as anomalous if it is tagged anomalous by at least 50% of users. In the Books dataset, ground truth anomalies are defined as nodes (with 21 features) having the tag *amazonfail*. Enron is a dataset of communication network with edges indicating email transmission between people. Each node contains 18 attributes describing metadata of the message (e.g., content length, number of recipients, etc.). Spammers are labeled as anomalous in this dataset. Enron has been extensively used as a benchmark for spam detection. In MADAN [13], these datasets are used in one of two formats: the original one, or a reduced version. The original dataset is the one commonly accessible, whereas the reduced version corresponds to the same graph-structure, but with a subset of the node features. We run our experiments on both versions of the datasets: the original one, and the reduced one. The statistics of all datasets are listed in Table 1.

	Books	Disney*	Enron*
Nodes	1,418	124	13,533
Edges	3,695	335	176,987
Features (original)	21	32	18
Features (reduced)	20	2	5
Anomalies	28	6	5

Table 1: Statistics of the three anomaly datection datasets in their two formats: original and reduced. The two datasets marked with * do not have a validation set with anomalies.

An important property of these datasets is that they do not define a validation set. In order to properly tune our hyperparameters, we create a validation set for Books by randomly splitting the set of normal nodes and anomalies into two subsets. The validation set corresponds to the first set of normal nodes and anomalies (composed of 695 normal nodes and 14 anomalies) and the rest is used as test set (the remaining 695 normal nodes and 14 anomalies). Validation sets were created for Disney and Enron, but only composed of normal nodes, due to the small number of anomalies (6 and 5, respectively). Details on the results and training on all datasets are given below.

Algorithms. We consider four existing Anomaly Detection Algorithms (ADAs). The first ADA is a simple One-Class Support Vector Machine (OCSVM) that separates anomalous nodes from the rest. The second ADA, called MEG [24] (Maximum Entropy Generators), is a generative-adversarial model that learns an energy function indicating the probability that a node is anomalous. Both MEG and OCSVM perform unsupervised anomaly detection on *unstructured* data: they only have access to the node features $\mathcal{X}$, and as such can only possibly detect feature-based anomalies.

In contrast, the two remaining ADAs have access to the graph structure $\mathcal{G}$ in addition to the node features $\mathcal{X}$: they could *a priori* detect the three forms of anomalies discussed in the introduction. MADAN [13] (Multi-scale Anomaly Detection on Attributed Networks), relies on multi-level community detection , while PAICAN [2] (Partial Anomaly Identification and Clustering in Attributed Networks) is a variational expectation-maximization approach learning the distribution of normal nodes.

Used on their own directly on un-augmented datasets, these four ADAs are the *baselines* to which we compare our approach. However, since our approach consists of augmenting datasets before applying an ADA (Figure 1), we also include the same four ADAs in our methods and results.

pre-training. The two models HideGNN and MaskGNN are self-supervised methods designed to reproduce the features of the nodes of the graph, and in doing so, conceptually learn what most non-anomalous nodes in the graph look like. These reproduced features are based on the graph structure and the node attributes, and we treat them as additional input for the anomaly detection models in order to augment the original dataset in various ways.

For the pre-training task, we perform a full exploration of the Cartesian product of the parameters within a certain range. Since we want our algorithm to transfer well from one dataset to another, we normalize the parameters when possible. For example, we define the number of masked entries in MaskGNN as a percentage rather than a given number. To normalize the learning rate given a set of other parameters (such as input dropout, weight decay, etc), we first experiment with a set of decreasing values for the learning rate (1, 0.1, 0.01, etc). For a given learning rate experiment, we stop training as soon as the model is *stable*, that is the loss does not diverge to infinity after 20 epochs; we record this learning rate λ_{stable} after 20 epochs. Given the hyperparameters $d_\lambda, d_L \in \mathbb{N}^*$ (which we tune in these experiments), we re-run our experiments by setting the learning rate to $\lambda_{\text{stable}}/d_\lambda$, record the loss L_{20} after 20 epochs, and use the early stopping criteria L_{20}/d_L. We use SGD optimizer and the losses defined in Equation (1) for HideGNN, and Equation (2) for MaskGNN.

Dataset augmentation. Once the pre-training task is over, we augment the original dataset by combining it with the output features reproduced by HideGNN and MaskGNN. Recall that this augmentation only affects the features of the graph and leaves edges intact. Let $\mathcal{X}$ and $\widehat{\mathcal{X}}$ be the original and reproduced features, respectively. We implement four types of node feature augmentations through the operator $\circledast$, which we treat as a hyperparameter. These augmentation methods are:

- *Replacement*: $\mathcal{X}$ is replaced by the reproduced features, that is $\mathcal{X} \circledast \widehat{\mathcal{X}} := \widehat{\mathcal{X}}$;
- *Concatenation*: $\mathcal{X}$ is replaced by the concatenation of the two sets of features, that is $\mathcal{X} \circledast \widehat{\mathcal{X}} := (\mathcal{X}, \widehat{\mathcal{X}})$;
- *Difference*: $\mathcal{X}$ is replaced by the difference between the two sets of features, that is $\mathcal{X} \circledast \widehat{\mathcal{X}} := \mathcal{X} - \widehat{\mathcal{X}}$;
- *Concatenation of the difference*: $\mathcal{X}$ is replaced by the concatenation of the original features and the difference between the two sets of features, that is $\mathcal{X} \circledast \widehat{\mathcal{X}} := (\mathcal{X}, \mathcal{X} - \widehat{\mathcal{X}})$.

Training and hyperparameters. Given that Books is the only dataset with a validation set, we tune the hyperparameters of the models on the Books validation set. For the different baseline algorithms, due to the small range of hyperparameter values (less than 100), we simply select the best result on the validation set and report the corresponding value on the test set. For HideGNN and MaskGNN, due to the large range of hyperparameter values (ranging from 10.000 to 12.000.000), we select the best 100 experiments and report the corresponding average value and standard deviation on the test set. We then use the same hyperparameters selected based on the Books validation set to obtain the corresponding results on the test sets of Disney and Enron.

Results. We evaluate model performance with the receiver operating characteristic Area Under the Curve (AUC), which corresponds to the likelihood of properly identifying the anomalous node between two nodes, one selected uniformly at random among the anomalous nodes and the other selected uniformly at random among the normal ones. In particular, an AUC smaller than 50% indicates that the performance of the given model is worse than random. We compare our method to the four ADAs described above, both before and after using our two pre-training algorithms HideGNN and MaskGNN. We summarize our results in Table 2.

As expected, the results are better and more stable on Books, since this is the only dataset with a proper validation set, which we use to tune the model hyperparameters. On this dataset, every case but one (applying MEG on the original dataset) is improved by applying our pre-training. Moreover, our method shows state-of-the-art results when combined with PAICAN on the reduced dataset. Apart from the best result, the table also shows significant improvements in the AUC when using our pre-training method, especially on OCSVM, but also on MADAN. This indicates that our algorithms improve previous techniques.

Recall from Table 1 that Disney and Enron respectively contain only 6 and 5 labeled anomalies: this is not enough to afford proper validation sets, which is why we transfer the hyperparameters tuned on Books to those two datasets. This situation also explains the high variability seen in Table 2 for Disney and Enron. It is difficult to get a clear message here: we report these numbers for the sake of transparency, and emphasize that more weight should be given to our results on Books.

It should be noted that the original MADAN paper [13] reports results for the reduced versions of the Books, Disney and Enron datasets of respectively 68, 93 and 66 percent. In that paper, they preferred a different approach to cope with the lack of proper validation sets: they try different values of `time_scale` and report the best results. This different approach explains the discrepancies with the corresponding numbers we report, namely 45.54, 19.49 and 73.64.

Overall, our two algorithms, HideGNN and MaskGNN, show promising results and could lead the way to further studies on anomaly detection in static attributed graphs.

5 Synthetic Dataset

To better understand the capabilities of our methods in finding anomalies, we perform further experiments with HideGNN and MaskGNN on a synthetic dataset called Thanksgiving. In this section, we present this dataset, discuss the results of our experiments, and demonstrate that pre-training with MaskGNN and HideGNN improves detection of anomalies coming from a combination of the features and the graph structure.

As discussed in the Introduction, anomalies in a graph-structured datasets can be grouped into three categories: feature-based, graph-based, and joint. Feature-based anomalies do not depend on the node neighborhood structure and can be detected by existing approaches that take into account only the sample features. Graph-based anomalies are only dependent on the connectivity structure of the node, and can be addressed using various exiting techniques that are clustering or community detection based. In this section, we are interested in joint anomalies, which depend on both the features of the nodes as well as their connectivity structure. To give an example of a joint anomaly, consider a

	Books		Disney*		Enron*	
	Original	Reduced	Original	Reduced	Original	Reduced
OCSVM†	35.90‡	37.33‡	36.44‡	**85.88**‡	55.90‡	43.22‡
Hide + OCSVM	49.58 (±1.47)	**52.21** (±2.41)	46.97 (±2.26)	43.16 (±1.03)	32.76 (±3.79)	**51.84** (±3.35)
Mask + OCSVM	**55.06** (±1.64)	48.49 (±2.07)	**47.63** (±1.73)	43.07 (±6.30)	**56.28** (±9.73)	50.46 (±6.55)
MEG† [24]	**63.12**‡	56.60‡	50.28‡	39.55‡	26.47‡	**63.08**‡
Hide + MEG	60.42 (±7.03)	**65.36** (±3.26)	46.51 (±6.51)	**69.44** (±3.47)	**44.22** (±14.05)	40.12 (±6.54)
Mask + MEG	57.06 (±6.55)	62.17 (±4.67)	**51.25** (±3.46)	62.44 (±13.38)	42.36 (±16.82)	40.63 (±12.21)
MADAN [13]	48.80‡	45.54‡	**67.51**‡	19.49‡	**61.81**‡	**73.64**‡
Hide + MADAN	**57.21** (±9.26)	52.08 (±7.64)	59.09 (±15.03)	**49.82** (±15.02)	54.24 (±14.41)	47.47 (±13.33)
Mask + MADAN	58.43 (±8.87)	**52.34** (±7.89)	55.98 (±14.07)	48.87 (±12.54)	60.55 (±14.58)	50.20 (±14.13)
PAICAN [2]	67.76‡	55.38‡	**75.14**‡	**73.45**‡	26.94‡	57.77‡
Hide + PAICAN	68.23 (±1.45)	**72.82** (±3.38)	62.73 (±8.14)	71.92 (±1.15)	**54.59** (±3.31)	58.65 (±10.07)
Mask + PAICAN	**68.90** (±1.15)	61.25 (±3.43)	67.31 (±6.87)	64.76 (±10.82)	45.92 (±8.08)	**65.40** (±5.46)

Table 2: The AUC score (in percent) of various anomaly detection methods applied to different datasets. Note that only Books has a validation set, so the models applied to it are the only ones whose hyperparameters were tuned. The results on Disney and Enron are based on the hyperparameters that worked well for Books. The two datasets marked with * do not have a validation set containing anomalies. The two algorithms marked with † do not use the graph structure in any way. The results marked with ‡ are expected to have a rather large error despite the lack of known confidence interval.

network composed of two communities of people, where every person (node) likes either dogs or cats (node features). In such a structure, if node v likes cats and is part of a community where all members like dogs, then v is an anomalous node. Liking either animal is not anomalous in itself, neither is being part of either of the two communities; however the combination of these two properties would render a node anomalous.

5.1 Thanksgiving Dataset

To test the ability of our methods in detecting joint anomalies, we create the Thanksgiving dataset. It consists of a graph with two communities generated by the Stochastic Block Model [17], where a few of the nodes in each community are joint anomalies (i.e., they are impossible to detect from the features alone or from the graph alone). The name of the dataset comes from the Thanksgiving dinner tradition, where families (some with different tastes or political convictions) travel home and reunite around the same dinner table.

Let $\mathcal{G} = (V, E)$ be a Thanksgiving graph with the set of nodes $V = V_1 \cup V_2$, such that V_1 and V_2 have roughly the same size. We define a negligible subset $V_A \in V$ of anomalous nodes such that $|V_A| \ll |V|$ and $|V_A \cap V_1| \simeq |V_A \cap V_2|$. Whether any pair of nodes $u, v \in V$ are connected by an edge is prescribed by the Stochastic Block Model [17], that is, the edge (u, v) exists with probability

$$p\big((u,v) \text{ is an edge}\big) = \begin{cases} p_{\text{in}} & \text{if } u, v \in V_1 \text{ or } u, v \in V_2 \\ p_{\text{out}} & \text{otherwise} \end{cases} .$$

This creates two separate communities of nodes that are clearly identifiable in the large graph limit as long as [33]

$$|V|(p_{\text{in}} - p_{\text{out}})^2 > 2(p_{\text{in}} + p_{\text{out}})\,. \qquad (3)$$

We next define the features of the nodes independently as

$$X_v[i] = \begin{cases} \mathcal{N}(0,1) & \text{if } v \in V_1 \setminus V_A \text{ or } v \in V_2 \cap V_A \\ \mathcal{N}(\mu,\sigma) & \text{otherwise} \end{cases} .$$

With this formulation, most nodes of V_1 have features with mean 0 and standard deviation 1, and most nodes of V_2 have features with mean μ and standard deviation σ. At the same time, a few anomalous nodes in V_1 have the features of V_2 and vice-versa. A representation of this dataset can be found in Figure 3.

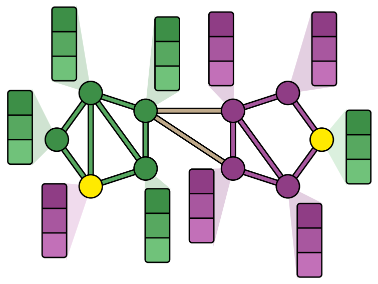

Figure 3: An example of the Thanksgiving dataset, with the anomalies highlighted in yellow. This dataset contains two communities of nodes, each having their own feature type. The purple and green edges are created with probability p_{in} while the brown edges are created with probability p_{out}. Anomalous nodes belong to one community in terms of graph structure, but have the features of the other community. In this example, the rightmost node is anomalous: it is connected to nodes of the purple community, but has features similar to that of the green community.

5.2 Thanksgiving Experiments

We test our models with the Thanksgiving dataset by independently varying the two main parameters of the data generation process: the mean μ of the node feature distribution for normal nodes in V_2 (and for anomalies in V_1), and the probability p_{out} parameterizing the number of connections between the two communities V_1 and V_2. We compare the results of five algorithms: OCSVM, HideGNN + OCSVM, MaskGNN + OCSVM, MADAN [13], and PAICAN [2]. We exclude MEG from our analysis because MEG, by design, is agnostic to network structure. We keep OCSVM as a trivial baseline to non-graph based classifiers.

In these experiments, we explore the conditions under which the different algorithms start to fail or succeed, based on the extent to which a node is anomalous. To successfully identify the anomalies in the Thanksgiving dataset, the model must discern (1) which of the two distributions the features of each node are sampled from, and (2) which of the two graph communities each node belongs to. We independently vary the difficulty of these tasks by changing (1) the difference between the feature means of the two communities ("easier" for higher μ when $\mu > 0$), and (2) the difference in edge density within and across communities (reported in terms of the number of inter-community connections; higher is "harder" up to 15, where $p_{\text{in}} = p_{\text{out}}$). We are thus less interested in the absolute performance of the models, and more interested in studying the *change* in results as the nodes become more anomalous, both in terms of features and in terms of connectivity structure.

For our experiments, we generate a set of 100 independent Thanksgiving variants for each of the two parameters we want to test, and apply the five algorithms listed above. Each dataset has 1000 nodes, out of which 20 are anomalous, with each node having 20 features. The expected number of links between two nodes of the same community is 15. For the experiments in Figure 4, the expected number of links between the two nodes across communities is 5, while the feature mean μ of the second community varies across the experiments. For the experiments in Figure 5, the feature distribution of the second community is the Normal distribution with mean $\mu = 3$ and variance $\sigma = 1$, while the inter-community edge probability p_{out} varies across experiments. The dataset augmentation method used to apply OCSVM to the outputs of HideGNN and MaskGNN is the difference, since it gave us the best results in the first phase of the experiments.

Experimenting with the feature mean. In the following set of experiments, we vary the feature mean of the second community in Thanksgiving and observe the performance of the various anomaly detection methods as the two feature means diverge. When the means of the communities are almost equal, both communities have similar features and the anomalous nodes become almost impossible to detect; as the means diverge, the joint anomalies become more evident. We start the experiments by setting the feature mean of the second community to zero (at which point there are no well-defined anomalies) and increasing its value to 10. Figure 4 summarizes the results of these experiments.

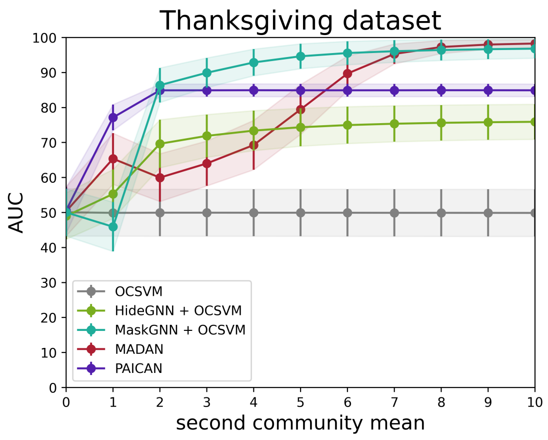

Figure 4: The evolution of the AUC (in percent) of five algorithms when modifying the mean of the Thanksgiving dataset. As the feature distribution means diverge, MaskGNN coupled with a basic classifier (OCSVM) quickly picks up the performance and beats the other algorithms, reaching almost an exact categorizations of the anomalies.

The results show that, when the features of the two communities are distinct-enough (at around mean 2), MaskGNN coupled with a simple classifier outperforms or compares to other methods. PAICAN seems to do relatively well in the case when the features of the anomalous and normal nodes are similar, but eventually plateaus as the difference between the two means increases.

Experimenting with the graph structure. Our second set of experiments explore the effect of varying the connectivity of the two communities in Thanksgiving. More precisely, we fix the number of edges inside a single community, but modify the connections between them, increasing from an average of 5 edges to 15 (which is the same as the expected number of edges within each community). Our goal is to see how the different algorithms perform when the structure of the graph changes, making the communities less distinct and more entangled. The results for these experiments are summarized in Figure 5.

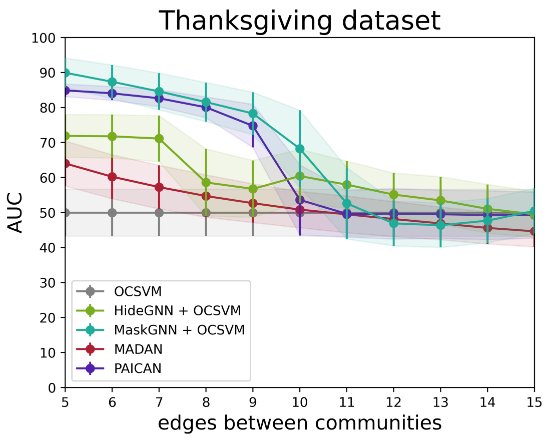

Figure 5: The evolution of the AUC (in percent) of five algorithms when modifying the number of connections between communities of the Thanksgiving dataset. In this case, our methods combined with the simple OCSVM always outperform or are comparable to MADAN and PAICAN. It is interesting to see that there is a strong change in behaviour around 10, which corresponds to the community detection threshold for the Stochastic Block Model [33].

The results of these experiments show that our models, combined with a simple OCSVM classifier, are comparable to, or even outperform, MADAN and PAICAN across all inter-community edge density. MaskGNN+OCSVM performs the best when the anomalies are clearly identifiable (in the range $5-9$). Note that when the number of connections between the communities reaches around 15 (i.e. when the expected number of edges within and across the communities are the same), then the two communities in Thanksgiving become unidentifiable and anomalies are no longer well-defined. It is thus expected that none of the algorithms in Figure 5 is able to correctly identify the nodes labeled anomalous in the dataset. Another interesting observation to note is the sudden drop in performance when the expected number of inter-community edges hits around 10. This, too, is expected and conforms to the theoretical results: for the Thanksgiving parameters used in these experiments (1000 nodes and 15 edges on average inside a community), the detection threshold, as defined in Equation (3), is around 10.

Interpretation. Although the Thanksgiving dataset is very simple compared to real-life datasets, these experiments give valuable insights on the performance and limitations of HideGNN and MaskGNN when labeled validation data is limited.

We remark the untapped opportunities for such methods to augment simple algorithms. Consider OCSVM, one of the simplest anomaly detection algorithms that is limited to feature-based anomalies. Combined with one of our pre-training methods, it becomes a serious contender for predicting graph-based and joint anomalies. In the case of the Thanksgiving dataset, this improvement often even surpasses more sophisticated algorithms, such as MADAN and PAICAN.

We also remark from Figure 4 and 5 that HideGNN+OCSVM and MaskGNN+OCSVM perform better in the range of "easier" anomalies ($2-4$ for the mean and $5-7$ for the number of connections) and do not significantly outperform other models when the anomalies are more subtle. This indicates that the decision to whether or not apply our methods might depend on the ultimate goal motivating the use of an anomaly detection algorithm. Indeed, if searching for subtle anomalies, then PAICAN might perform better, whereas MaskGNN+OCSVM shows stronger results on Thanksgiving in identifying clear joint anomalies.

6 Conclusion and Further Discussion

Unsupervised anomaly detection remains a difficult problem to solve. Its challenges lie in the ambiguity (and often, subjectivity) of what constitutes an anomaly, coupled with the fact that such instances are generally so rare that enough examples of anomalies are hard to find. As such, supervised machine learning models are not a good fit for solving such problems.

In this paper, we explore self-supervised methods on graph-structured data to learn what *normal* nodes look like for a given dataset. In our approaches, the graph structure defines normality with respect to a local neighborhood of the sample. The results summarized in Table 2 and Figures 4 and 5 show that pre-training on a dataset with either of the self-supervised learning methods HideGNN and MaskGNN offers promising results in helping with the anomaly detection task in graph datasets. With proper tuning of the hyperparameters (the results on Books that are obtained by turning on a validation set), both our approaches show an overall superior performance to existing methods. For joint anomalies, which are specific to graph datasets, our methods outperform existing baselines when there is a clear-enough distinction between the distribution of the features of the anomalies or the community graph structures. This is true even as our pre-training is combined with a basic anomaly detection scheme such as OCSVM.

One of the interesting aspects of this work is the way in which pre-training is combined with other algorithms. When augmenting datasets with the output of a learning algorithm, the common practice is to combine the embeddings (e.g., vectors) learned by the hidden layers (usually the ones close to the output layer) with existing features. In this work, we use the prediction (output layer) of the model to augment the dataset. Our results reinforce the idea that the reproducing scheme of

HideGNN and MaskGNN has potential when it comes to anomaly detection. The results of Figure 4 and 5, combined with the ones in Table 2, show that our models are powerful at identifying anomalies in various settings, since most anomalies will have some kind of unexpected behaviour arising from their features, their connections, or a combination of both.

Overall, the approach proposed in this work, which consists of combining a GNN model with hiding or masking methods, shows promising results and could lead to further development in the field of structural data. Such methods could also be used in dynamical structured data (temporal graphs), or for tasks beyond anomaly detection, such as node classification or edge prediction.

References

[1] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688, 2015.

[2] Aleksandar Bojchevski and Stephan Günnemann. Bayesian robust attributed graph clustering: Joint learning of partial anomalies and group structure. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[3] Carlos Castillo, Debora Donato, Aristides Gionis, Vanessa Murdock, and Fabrizio Silvestri. Know your neighbors: Web spam detection using the web topology. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 423–430, 2007.

[4] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *CoRR*, abs/1901.03407, 2019.

[5] Tong Che, Ruixiang Zhang, Jascha Sohl-Dickstein, Hugo Larochelle, Liam Paull, Yuan Cao, and Yoshua Bengio. Your GAN is secretly an energy-based model and you should use discriminator driven latent sampling. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[6] Guillem Cucurull, Perouz Taslakian, and David Vazquez. Context-aware visual compatibility prediction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[7] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2702–2711. JMLR.org, 2016.

[8] Rajendra Kumar Dwivedi, Arun Kumar Rai, and Rakesh Kumar. Outlier detection in wireless sensor networks using machine learning techniques: a survey. In *2020 International Conference on Electrical and Electronics Engineering (ICE3)*, pages 316–321. IEEE, 2020.

[9] Bahare Fatemi, Perouz Taslakian, David Vázquez, and D. Poole. Knowledge hypergraphs: Prediction beyond binary relations. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2019.

[10] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 2017.

[11] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[12] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014.

[13] Leonardo Gutiérrez-Gómez, Alexandre Bovet, and Jean-Charles Delvenne. Multi-scale anomaly detection on attributed networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):678–685, Apr. 2020.

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[15] Ayush Hariharan, Ankit Gupta, and Trisha Pal. Camlpad: Cybersecurity autonomous machine learning platform for anomaly detection. In *Future of Information and Communication Conference*, pages 705–720. Springer, 2020.

[16] Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[17] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[18] Luca Invernizzi, Paolo Milani Comparetti, Stefano Benvenuti, Christopher Kruegel, Marco Cova, and Giovanni Vigna. Evilseed: A guided approach to finding malicious web pages. In *2012 IEEE symposium on Security and Privacy*, pages 428–442. IEEE, 2012.

[19] Bernard J Jansen. Click fraud. *Computer*, 40(7):85–86, 2007.

[20] Wei Jin, Tyler Derr, Haochen Liu, Yiqi Wang, Suhang Wang, Zitao Liu, and Jiliang Tang. Self-supervised learning on graphs: Deep insights and new direction. *arXiv preprint arXiv:2006.10141*, 2020.

[21] P Naga Jyothi, D Rajya Lakshmi, and KVSN Rao. A supervised approach for detection of outliers in healthcare claims data. *Journal of Engineering Science & Technology Review*, 13(1), 2020.

[22] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[23] Nir Kshetri. The economics of click fraud. *IEEE Security & Privacy*, 8(3):45–53, 2010.

[24] Rithesh Kumar, Anirudh Goyal, Aaron C. Courville, and Yoshua Bengio. Maximum entropy generators for energy-based models. *CoRR*, abs/1901.08508, 2019.

[25] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[26] Kyumin Lee, James Caverlee, and Steve Webb. Uncovering social spammers: social honeypots+ machine learning. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 435–442, 2010.

[27] Yixin Liu, Shirui Pan, Ming Jin, Chuan Zhou, Feng Xia, and Philip S Yu. Graph self-supervised learning: A survey. *arXiv preprint arXiv:2103.00111*, 2021.

[28] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254, 2009.

[29] Franco Manessi and Alessandro Rozza. Graph-based neural network models with multiple self-supervised auxiliary tasks. *Pattern Recognition Letters*, 2021.

[30] Jorge Meira, Rui Andrade, Isabel Praça, João Carneiro, Verónica Bolón-Canedo, Amparo Alonso-Betanzos, and Goreti Marreiros. Performance evaluation of unsupervised techniques in cyber-attack anomaly detection. *Journal of Ambient Intelligence and Humanized Computing*, 11(11):4477–4489, 2020.

[31] Vangelis Metsis and et al. Spam filtering with naive bayes – which naive bayes? In *THIRD CONFERENCE ON EMAIL AND ANTI-SPAM (CEAS*, 2006.

[32] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4602–4609. AAAI Press, 2019.

[33] Elchanan Mossel, Joe Neeman, and Allan Sly. A proof of the block model threshold conjecture. *Combinatorica*, 38(3):665–708, 2018.

[34] Emmanuel Müller, Patricia Iglesias Sánchez, Yvonne Mülle, and Klemens Böhm. Ranking outlier nodes in subspaces of attributed graphs. In *2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW)*, pages 216–222, 2013.

[35] Myle Ott, Claire Cardie, and Jeff Hancock. Estimating the prevalence of deception in online review communities. In *Proceedings of the 21st international conference on World Wide Web*, pages 201–210, 2012.

[36] Zhen Peng, Yixiang Dong, Minnan Luo, Xiao-Ming Wu, and Qinghua Zheng. Self-supervised graph representation learning via global context prediction. *arXiv preprint arXiv:2003.01604*, 2020.

[37] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, Nagendra Modadugu, et al. The ghost in the browser: Analysis of web-based malware. *HotBots*, 7:4–4, 2007.

[38] Meng Qu, Yoshua Bengio, and Jian Tang. Gmnn: Graph markov neural networks. In *International Conference on Machine Learning*, pages 5241–5250, 2019.

[39] I Sadgali, N Sael, and F Benabbou. Performance of machine learning techniques in the detection of financial frauds. *Procedia computer science*, 148:45–54, 2019.

[40] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, volume 10843 of *Lecture Notes in Computer Science*, pages 593–607. Springer, 2018.

[41] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[42] Yaochen Xie, Zhao Xu, Zhengyang Wang, and Shuiwang Ji. Self-supervised learning of graph neural networks: A unified review. *arXiv preprint arXiv:2102.10757*, 2021.

[43] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 974–983. ACM, 2018.

[44] KiJung Yoon, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard S. Zemel, and Xaq Pitkow. Inference in probabilistic graphical models by graph neural networks. In *53rd Asilomar Conference on Signals, Systems, and Computers, ACSCC 2019, Pacific Grove, CA, USA, November 3-6, 2019*, pages 868–875. IEEE, 2019.

[45] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3391–3401, 2017.

[46] Jiaqi Zeng and Pengtao Xie. Contrastive self-supervised learning for graph classification. *arXiv preprint arXiv:2009.05923*, 2020.

[47] Shuangfei Zhai, Yu Cheng, Weining Lu, and Zhongfei Zhang. Deep structured energy based models for anomaly detection. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1100–1109. JMLR.org, 2016.