# GAWD: Graph Anomaly Detection in Weighted Directed Graph Databases

Meng-Chieh Lee
Carnegie Mellon University
mengchil@cs.cmu.edu

Hung T. Nguyen
Princeton University
hn4@princeton.edu

Dimitris Berberidis
Carnegie Mellon University
dbermper@andrew.cmu.edu

Vincent S. Tseng
National Chiao Tung University
vtseng@cs.nctu.edu.tw

Leman Akoglu
Carnegie Mellon University
lakoglu@andrew.cmu.edu

***Abstract*—Given a set of node-labeled directed weighted graphs, how to find the most anomalous ones? How can we summarize the normal behavior in the database without losing information? We propose GAWD, for detecting anomalous graphs in directed weighted graph databases. The idea is to (1) iteratively identify the "best" substructure (i.e., subgraph or motif) that yields the largest compression when each of its occurrences is replaced by a super-node, and (2) score each graph by how much it compresses over iterations — the more the compression, the lower the anomaly score. Different from existing work [1] on which we build, GAWD exhibits ($i$) a *lossless* graph encoding scheme, ($ii$) ability to handle numeric edge weights, ($iii$) interpretability by common patterns, and ($iv$) scalability with running time linear in input size. Experiments on four datasets injected with anomalies show that GAWD achieves significantly better results than state-of-the-art baselines.**

## I. Introduction

Given a large graph database containing directed weighted node-labeled graphs, how can we detect the anomalous graphs? Many studies succeed in detecting anomalies but fail to give satisfying interpretations. This raises another prominent problem — how can we spot anomalies and summarize the normal behavior without simultaneously losing information?

In recent years, graph [2], [3] and node embedding [4] have attracted a lot of attention. These methods have been used in anomaly detection in conjunction with off-the-shelf anomaly detectors. Embedding-based models, however, lack interpretability. In contrast, structure-based methods enable domain experts to conduct post-analysis to reveal root causes of anomalies. Several structure-based methods [5], [6] detect anomalies by compressing graphs with a substructure that yields the largest compression. The selected substructure is replaced by a super-node and the process continues in iterations. As a result, graphs with more common substructures (and hence compress more) are deemed less anomalous than those with fewer substructures.

However, neither embedding- nor structure-based methods are perfect: (1) both of these methods cannot totally avoid information loss, which causes difficulty in interpreting results, and (2) none of the structure-based methods can handle weighted graphs, which prevents them from detecting anomalies caused by edge weights.

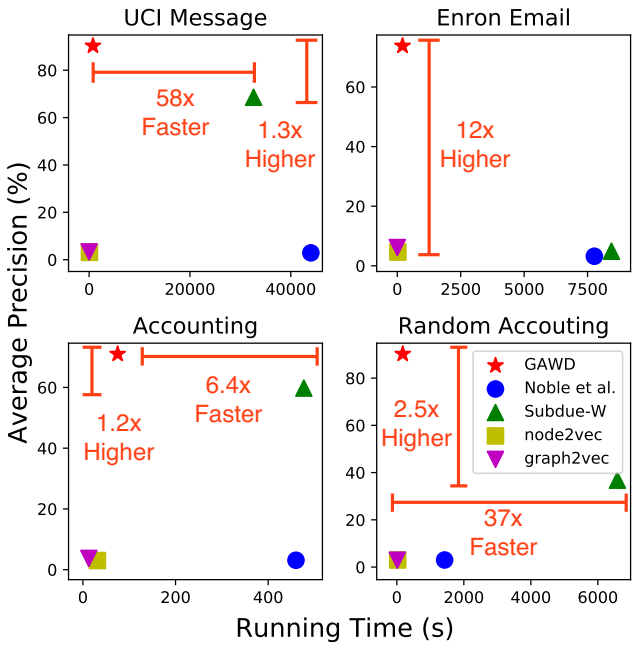



Fig. 1: **GAWD wins on both effectiveness and scalability:** We evaluate four datasets and show the big gap between it and competitors w.r.t. average precision and run time.

Here we propose GAWD to address the aforementioned problems. In a nutshell,

- **Lossless encoding:** GAWD builds on Noble and Cook [5] in terms of identifying frequent subgraphs and compressing the graphs in the input database by replacing each of its occurrences by a super-node. This results in a loss of connectivity information for nodes outside the substructure that are connected to nodes within. We address this issue by incorporating "rewiring" information into our encoding, such that the compressed graph can be reconstructed into the original graph *losslessly*.
- **Handling Weighted Graphs:** We propose a novel encoding scheme for handling numeric edge weights. Given a substructure we estimate a "representative" weight for its

edges, as well as extend the encoding of a compressed graph to incorporate corrections for true weights such that decompression can be done losslessly.
- **Interpretability and Scalability:** The (lack of) frequent subgraphs common in the database provide a means to explain anomalousness. Moreover, GAWD exhibits linear scalability in the input size.

As shown in Figure 1, experimental results on four real-world datasets with injected anomalies show that GAWD provides better trade-off between detection performance and running time compared to both existing graph embedding- and structure-based methods. Moreover, GAWD is lossless in contrast with lossy compression of existing methods [1], [5], [6] and, therefore, able to backtrack the process after compression, while other lossy methods lose those information after compression.

The rest of this paper is organized as follows: we briefly review related literature in Section 2. In Section 3 we introduce the problem statement and detail our proposed method in Section 4. Experimental results are presented in Section 5. Section 6 concludes the work.

## II. Related Work

Being one of the closest real-world applications, anomaly detection has drawn a lot of attention from academics [7]–[11]. Many anomaly detectors have been developed, such as LOF [12], Isolation Forest [13] and LODA [14]. Some traditional machine learning methods such as kNN [15] and PCA [16] can be used as an anomaly detector as well. To this reason, a useful toolkit for this field is also developed [17]. These detectors can be easily used with feature-based or embedding-based methods to achieve effective results. Our work introduces an anomaly detection for graphs approach, while using popular graph embedding methods (with anomaly detectors) as baselines. We will discuss them both in this section.

### A. Indirect Approaches via Graph Embedding

Graph embedding has been widely studied in the last decade. One reason for its popularity is its flexibility concerning downstream applications. Graph embedding can be used to detect anomalous graphs in conjunction with off-the-shelf anomaly detectors.

One of the most famous branches is node embedding, where each node in the graph is mapped to low dimensional space. node2vec [4] could highly identify graph structures with biased random walks using BFS and DFS. GraphSAGE [2] increases the generalizability to the unseen nodes by training the aggregator function by node features. A simple way to extend node embedding to graph embedding is to sum all node vectors up, which has been widely used as a baseline in graph embedding approaches. Unlike node embedding, graph embedding aims to directly map each graph in the graph database into a vector. graph2vec [3] uses document embedding neural networks to embed node-labeled graphs. Node embedding methods could also be used in graph embedding by averaging the embedding of nodes. Variational graph autoencoder [18] encodes the graph by a two-layer graph convolutional network and decodes by an inner-product decoder.

However, few graph embedding methods could handle node labels and edge weights at the same time. An additional drawback of graph embedding is the low interpretability of the representations, and as a result, anomalousness.

TABLE I: **GAWD matches all specs**, while competitors miss one or more of the desired properties. * implies the method can only accept (categorical) edge labels—weights need to be discretized.

| Property \ Method | node2vec [4] | graph2vec [3] | Noble *et al.* [5] | GBAD [6] | OddBall [19] | Yagada [20] | GAWD |
|---|---|---|---|---|---|---|---|
| **Handle Graph Database** | ✔ | ✔ | ✔ | ✔ | | ✔ | ✔ |
| **Handle Node Labels** | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| **Handle Edge Weights** | ✔ | | | | ✔ | ✔* | ✔ |
| **Lossless** | | | | | | | ✔ |
| **Scale Linearly** | ✔ | ✔ | | | ✔ | | ✔ |

### B. Direct Anomaly Detection for Graphs

In order to seek higher interpretability, we turned to graph-based anomaly detection. Anomaly detection has been studied extensively for its applicability to real-world scenarios. Careful scrutiny of these studies can be found in [21].

On the one hand, some studies try to spot the graph anomalies by mining graphs' structural features. OddBall [19] detects anomalous nodes in a *single* weighted graph, but does not extend to graph databases. ReFeX [22] includes recursive features to extract information even beyond direct neighbors. To increase the interpretability, features are analyzed in pairs in [23], which can be easily visualized and point out the outliers. Moreover, LookOut [24] further turns the anomaly detection into a 2-dimension plots selection problem, and picks up the most explainable plots to the anomalies.

On the other hand, some researchers seek to explain the graph anomalousness by frequent patterns among graphs. Cook *et al.* [1] proposed a graph substructure discovery framework, which Noble and Cook [5] leverage in anomaly detection by using compression rates in each iteration. Eberle *et al.* [6] detect unexpected structural deviations, defined as frequent patterns with slight changes. These structure-based methods do not take edge weights into consideration. For numerical weights, Yagada [20] uses discretization to assign edges with discrete labels. However, discretization loses information and underperforms as we demonstrate in our experiments.

Nevertheless, none of the above methods fulfills all the specs of GAWD. Table I contrasts GAWD against the existing state-of-the-art.

## III. Problem Definition and General Framework

We consider a graph database consisting of $I$ *node-labeled directed weighted graphs* $\mathcal{G} = \{G_1(V_1, E_1), \ldots, G_I(V_I, E_I)\}$,

TABLE II: Notations used in the paper.

| Symbols | Definitions |
|---|---|
| $\mathcal{G}$ | Graph database |
| $G_i$ | $i$-th graph in the database |
| $I$ | Number of graphs in database |
| $J$ | Total number of iterations in our algorithm |
| $V_i$ | Node set of $i$-th graph |
| $E_i$ | Edge set of $i$-th graph |
| $\mathcal{T}$ | Set of unique node labels |
| $t(v)$ | Label of node $v$ |
| $w(u,v)$ | Edge weight of edge $(u,v)$ |
| $P_j$ | Substructure found in iteration $j$ |

where each graph $G_i(V_i, E_i)$ has a set of labeled nodes $V_i$ and a set of weighted edges $E_i$. For each node $v \in V_i$, $t(v) \in \mathcal{T}$ denotes the label of node $v$, where $\mathcal{T}$ represents the set of unique node labels, e.g., types of accounts in a company. Each edge $(u,v) \in E_i$ is associated with a weight $w(u,v)$, e.g., number of transactions between 2 accounts. Table II defines a comprehensive list of our notations.

### A. Problem Definition

Our anomaly detection problem is concisely defined as follows:

**Definition 1** (Anomaly Detection in Directed Weighted Graph Database)**.** *Given a node-labeled directed weighted graph database $\mathcal{G} = \{G_1(V_1, E_1), ..., G_I(V_I, E_I)\}$, compute anomaly scores $a_i$ for each graph $G_i \in \mathcal{G}$.*

### B. General Framework

Our method follows a general information-theoretic framework depicted in Algorithm 1. This framework generalizes previous graph anomaly detection methods, i.e., [1], [5]. Given a graph database, the idea is to iteratively identify the "best" substructure that yields the largest compression, replacing each of its occurrences with a super-node. Each graph in the database is then scored by how much it compresses over iterations—the more the compression, the lower the anomaly score. In Algorithm 1, blue texts pinpoints the differences in GAWD compared to those in [1], [5].

In particular, the existing method in [5] detects the frequent patterns in line 2 by beam search. To identify the best pattern, they use the one that can minimize the total description length of graphs in the database in line 6. However, they only take the compressed node and edge information into consideration (lossy encoding). They then compress the graphs by that pattern in line 7. This process will keep running until no more pattern is found. The heart of their approach is the encoding scheme of graphs by Minimum Description Length (MDL) principle, which includes encoding the structure of graphs, i.e., nodes and edges, as follows:

The total encoding length for nodes is:

$$\text{vbits}(G_i) = \log^* |V_i| + |V_i| \log_2 |\mathcal{T}|, \quad (1)$$

**Data:** A graph database
**Result:** Anomaly scores for all graphs
1 **while** *True* **do**
2  Detect frequent patterns in graph database;
3  **if** *No pattern is found* **then**
4  Break;
5  **end**
6  Identify the pattern which can compress the graphs in database the most;
7  Compress the graphs by this pattern;
8 **end**
9 Compute the anomaly scores by compression rate;

**Algorithm 1:** General Framework for Graph Anomaly Detection (followed by [1], [5], blue text points out the differences with GAWD)

where $|\mathcal{T}|$ denotes the number of unique node labels in $G_i$. $\log^*$ is the universal code length used to encode the numeric value. We first need $\log^* |V_i|$ bits to encode the number of nodes, and then need $\log_2 |\mathcal{T}|$ bits to encode the label for each node.

The total encoding length for the adjacency matrix is:

$$\text{rbits}(G_i) = \log^* b + \sum_{p=1}^{|V_i|} \log_2 (b+1) + \log_2 \binom{|V_i|}{k_p}, \quad (2)$$

where $b$ denotes the highest out-degree in $G_i$, and $k_p$ denotes the particular out-degree of $p^{\text{th}}$ node. $\log^* b$ bits are needed to encode the highest out-degree. For each row of adjacency matrix, $\log_2 (b+1)$ bits are needed to encode the degree of node. Given $k_p$ as the number of 1(s) occurring in $p^{\text{th}}$ row, we know that there are only $\binom{|V_i|}{k_p}$ possible permutations, so we need $\log_2 \binom{|V_i|}{k_p}$ bits to encode the positions of 1(s) in the $p^{\text{th}}$ row.

The total encoding length for edges is:

$$\text{ebits}(G_i) = \log^* m + |E_i| \log_2 m, \quad (3)$$

where $m$ denotes the largest edge weight. We first need $\log^* m$ bits to encode the largest edge weight, and then $\log_2 m$ bits to encode the weight for each edge.

Thus, the total encoding length for $G_i$ in $j$-th iteration is:

$$DL(G_i(j)) = \text{vbits}(G_i(j)) + \text{rbits}(G_i(j)) + \text{ebits}(G_i(j)), \quad (4)$$

Different from [5], in GAWD, we replace the beam search in line 2 by *gSpan*, which is a much faster subgraph mining technique; we design a novel graph encoding scheme, being used in line 6, which accepts edge weight (described in Section IV-A) and is lossless (described in Section IV-B).

## IV. Proposed Method - GAWD

Next we provide the details of GAWD. Given a substructure $P_j = (V_j, E_j)$ at iteration $j$, which is a node-labeled simple graph, the first task is to identify a "representative" weight for edge $(u,v) \in E_j$, denoted $w^*_{P_j}(u,v)$. Given the edge-weighted $P_j$, our encoding scheme involves:

1) encoding $P_j$,
2) encoding each compressed graph $\overline{G}_i = (\overline{V}_i, \overline{E}_i)$ resulted from replacing each occurrence/instance of $P_j$ (ignoring edge weights) in $G_i$ with a super-node, and
3) encoding auxiliary information for lossless reconstruction of $G_i$, given $P_j$ and $\overline{G}_i$.

Steps (1) and (2) use the encoding scheme in Subdue [1], the details of which we omit due to space limit. Here we describe our novel contributions in Step (3), specifically, weight and rewiring encoding, respectively $(i)$ handling edge weights and $(ii)$ enabling lossless reconstruction. Total compression cost (or description length) of the graph database is the sum of bits used for encoding (1)–(3).

### A. Weight Encoding

*1) Representative Weight Discovery:* Given a substructure $P_j$, the representative edge weight $w^*_{P_j}(u,v)$ for each edge $(u,v) \in E_j$ need to be identified before evaluating the substructures toward compression. Let $E_{j,(u,v)}$ denote all the edges in the instances of substructure $P_j$ in the database corresponding to $(u,v) \in E_j$. We turn this into an optimization problem based on the Minimum Description Length (MDL) encoding. Given a candidate weight $w$, we denote the bits needed to correct with respect to the true weight of an edge instance $(s,t) \in E_{j,(u,v)}$ by $L(w, w_{P_j}(s,t))$ (details in Section IV-A2). The optimization problem is then formulated as:

$$w^*_{P_j} = \min_{w} \sum_{(s,t)\in E_{j,(u,v)}} L(w, w_{P_j}(s,t)) \tag{5}$$

While not convex, the optimization in (5) is only 1-dimensional and hence relatively easy to solve. We employ Dichotomous Search [25], which returns the optimal solution in most cases. It efficiently takes only $O(|E_{j,(u,v)}| \log_2 R)$, where $R$ is the numeric search range of weights.

*2) Weight Corrections:* After discovering $w^*_{P_j}$, we now encode the weights in each instance. For each super-node $s \in \overline{V}_i$ of $\overline{G}_i$, we denote by $g_s = (V_s, E_s)$ the substructure instance in $G_i$ corresponding to $s$, which is isomorphic to $P_j$ in structure. For each edge $(u,v) \in E_s$, we encode its weight correction using:

$$L(w,w') = \begin{cases} 1 \text{ bit}, & \text{if } w - w' = 0 \\ 2\log_2(|w-w'|) + 3 \text{ bits}, & \text{otherwise} \end{cases},$$

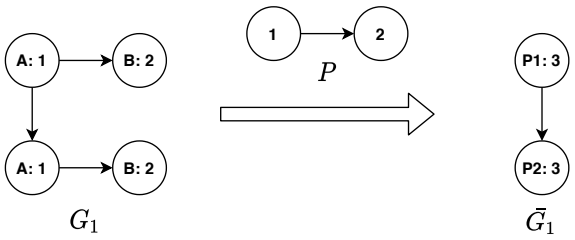
where $w = w^*_{P_j}(u,v)$ and $w' = w_{g_s}(u,v)$. 1 bit is used to identify whether the weight correction is needed. If so, an extra 1 bit is used to record the sign of the error. $2\log_2(|w-w'|)+1$ bits is used to encode the numeric value by universal code.

We remark that instead of discretizing edge weights into labels, our encoding scheme handles the numeric value and is lossless. Thus, the total encoding length of weight corrections is:

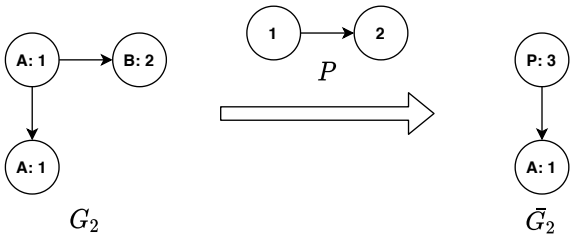
$$\text{mbits}(G_i) = \sum_{s\in\overline{V}_i} \sum_{(u,v)\in E_s} L(w, w') \tag{6}$$

### B. Rewiring Encoding

After replacing $P_j$ with a super-node, all the edges connected to $P_j$ merge into super-edges. (Weight of a super-edge $e = (x,y) \in \overline{E}_i$, denoted $w_{\overline{G}_i}(x,y)$, is the sum of the weights of all edges that it represents.) For lossless reconstruction, the edge (re)connectivity information needs to be encoded. There are two possible cases: (1) both $x$ and $y$ are super-nodes corresponding to non-overlapping instances of $P_j$, and (2) only one of them is a super-node.




(a) Case 1: A super-edge is created between two super-nodes after compression.




(b) Case 2: A super-edge is created between one super-node and one regular node after compression.

Fig. 2: **Rewiring Encoding:** Two possible cases that the edge (re)connectivity information are different.

For the former case, we first encode the cardinality of $e$, denoted $c_e$, depicting how many edges it represents, using:

$$L(c_e) = \log_2(|V_j|^2) = 2\log_2(|V_j|) \text{ bits} . \tag{7}$$

For *each* edge, we encode substructure node IDs of its source and its destination using $2\log_2(|V_j|)$ bits total, and then encode its weight using $\log_2(w_{\overline{G}_i}(x,y))$ bits.

For the latter case, w.l.o.g. let $x$ be the super-node. We encode how many edges $e$ branches to, denoted $b_y$, using:

$$L(b_y) = \log_2(|V_j|) \text{ bits} . \tag{8}$$

In other words, $b_y$ denotes how many distinct nodes within $g_x$ that $y$ connects to. For each edge we encode the substructure node ID of $y$'s neighbor $n \in V_x$ using $\log_2(|V_j|)$ bits. We then encode the weight of each edge the same as in the former case.

The total encoding length for rewiring is:

$$\begin{aligned} \text{wbits}(G_i) &= \textstyle\sum_{e=(x,y)\in\overline{E}_i} (|e|-1)\log_2(w_{\overline{G}_i}(x,y)) \\ &+ \begin{cases} (|e|+1)L(c_e) & \text{if } x,y \text{ are both super nodes} \\ (|e|+1)L(b_y) & \text{if } x \text{ is super node} \end{cases} \end{aligned} \tag{9}$$

where $e = (x,y)$ denotes the super-edge connecting from node $x$ to node $y$ in compressed graph $\overline{G}_i$, and $|e|$ denotes the multiplicity of the super-edge.

**Data:** A database $\mathcal{G} = \{G_1, ..., G_I\}$, min support range $(ms_{\max}, ms_{\min})$, decay rate $d$ (i.e., 0.9 by default.)
**Result:** Anomaly scores $a = \{a_1, ..., a_I\}$ for all graphs in $\mathcal{G}$
1 initialization: $ms = ms_{\max}$; $j = 0$;
2 **while** $ms \geq ms_{\min}$ **do**
3 $\quad \mathcal{P}_j = gSpan(\mathcal{G}, ms)$;
4 $\quad$ Discover $w^*_{P_j}(u, v)$ for all $P_j \in \mathcal{P}_j$ (See § IV-A1);
5 $\quad$ Identify $P^*_j \in \mathcal{P}_j$ yielding largest (positive) compression;
6 $\quad$ **if** *no* $P^*_j$ *is found* **then**
7 $\quad\quad ms := ms * d$;
8 $\quad\quad$ Continue;
9 $\quad$ **end**
10 $\quad$ Compress $\mathcal{G}$ by $P^*_j$ and save $c^j_i$ in (11) for all $G_i \in \mathcal{G}$;
11 $\quad j := j + 1$;
12 **end**
13 $a_i = 1 - \frac{1}{j}\sum_{k=1}^{j}\left[(j-k+1) * c^k_i\right]$, for all $G_i \in \mathcal{G}$;

**Algorithm 2:** GAWD-Anomaly Scoring

### C. Overall Algorithm

The total encoding length of graph $G_i$ in $j$-th iteration is given as the following:

$$DL^*(G_i(j)) = DL(G_i(j)) + \text{mbits}(G_i(j)) + \text{wbits}(G_i(j)). \tag{10}$$

Algorithm 2 gives the steps of GAWD. We use $gSpan$ [26] for frequent substructure mining in Line 3. In Lines 4-6, we search for the best weighted substructure yielding the largest compression. To improve the efficiency, we gradually decrease minimum support from maximum value to minimum in Lines 7-9, if there is no substructure found. Once we identify the best substructure $P_j$ in iteration $j$, we compress the graphs in the database by $P_j$ and save the compression rate $c^j_i$, for each graph $i$, defined as:

$$c^j_i = \frac{DL^*_{j-1}(G_i) - DL^*_j(G_i)}{DL^*_0(G)}, \tag{11}$$

where $DL_j(G_i)$ is the description length of $G_i$ after $j$ iterations. Finally, we compute the anomaly scores in Line 13. The anomaly score ranges from 0 to 1, where 1 means the most anomalous and 0 means the least anomalous. The compression rates are linearly weighted by the term $j-k+1$, where it means that the earlier we identify the substructure as the best one, the less anomalous that the graphs containing it are.

### D. Complexity Analysis

**Lemma 1.** GAWD *is linear on the input size, taking time* $O(nI|E|\log|V|)$*, where* $n$ *denotes the number of frequent substructures,* $I$ *denotes the number of graphs, and* $|V|$ *and* $|E|$ *denote the average numbers of nodes and edges.*

*Proof.* In the worst case, $n$ frequent substructures are detected by $gSpan$ and all can be used for compression with no conflict, then GAWD at most will iterate $n$ times. Moreover, $O(I|E|\log|V|)$ is the time complexity of $gSpan$, where $I$ denotes the number of graphs in graph database, $|E|$ denotes the average edge number of graphs, and $|V|$ denotes the average node number. The Dichotomous Search is also efficient, taking $O(|E_{j,(u,v)}|\log_2 R)$, where $E_{j,(u,v)}$ denotes all the edges in the instances of substructure $P_j$, and $R$ denotes the numeric search range of weights. For compression, it takes $O(I|E|)$ to redirect edges for each graph. The complexity of GAWD is $O(n(|E_{j,(u,v)}|\log_2 R + I|E| + I|E|\log|V|))$. Empirically, $|E_{j,(u,v)}|$ and $\log_2 R$ are small constant values which are negligible. Therefore, the complexity is $O(nI|E|\log|V|)$. ■

## V. Experiments

We design experiments to answer the following questions:

Q1. **Effectiveness**: How well does GAWD work on anomaly detection?

Q2. **Scalability**: How does GAWD's running time grow with input size?

Our code and injected datasets along with labels (except Accounting Dataset due to privacy issue) are made publicly available[1]. Experiments are run on a machine with 3.2GHz CPU and 256 GB RAM.

### A. Settings

*1) Datasets:* We use four datasets illustrated in Table III. The detailed description of all datasets are shown as follows:

- **UCI Message Dataset [27]:** This recorded the communications between students at UCI where nodes and edges denote students and messages respectively. To capture the role information, we adopt role2vec [28] to embed nodes in the complete graph, and use the 10 groups clustered by Agglomerative Clustering as the node labels. The data is split into hours to form a graph database.
- **Enron Email Dataset [29]:** This contains the emails passing between colleagues in Enron Company from 2000 to 2002. We assign the job positions to each employee as node labels. The data is split into day communication graphs to form a graph database.
- **Accounting Dataset:** This is from an anonymous institution, containing accounts (nodes) and transactions (edges) that precisely reflect the money flow between company accounts. Each graph captures a set of transactions within a unique expense report.
- **Random Accounting Dataset:** Since the accounting dataset is proprietary, we generate a synthetic database with generated graphs following the same statistical characteristics as in the accounting graphs. The generation process details are described in the appendix for clarity.

We treat edge multiplicities as weights for all four graph databases. There are no ground truth anomalies in all the databases. To evaluate effectiveness, we inject anomalies into randomly sampled 3% of the graphs in each database as [21] suggested, which had also been done by several other studies [30], [31]. For each sampled graph, we (i) randomly select an edge $(u, v) \in E_i$, and (ii) multiply its weight $w(u, v)$ by $10^d$, where $d$ denotes the digit count of the upper fence in the boxplot of weights for $(t(u), t(v))$ edges. This aims to simulate unusual behaviors between the users or accounts.

[1] https://github.com/mengchillee/GAWD/tree/master/data

| Method | Precision @45 | Precision @90 | Recall @45 | Recall @90 | AUC | AP | Time |
|---|---|---|---|---|---|---|---|
| node2vec | 6.7 | 5.6 | 3 | 5.1 | 47.6 | 3.1 | 58.6s |
| graph2vec | 2.2 | 1.1 | 1.0 | 1.0 | 48.7 | 3.1 | 2.9s |
| Noble *et al.* | 0.0 | 0.0 | 0.0 | 0.0 | 47.6 | 3.1 | 43988s |
| Subdue-W | **100.0** | 60.0 | **45.5** | 54.5 | **94.8** | 68.6 | 32621s |
| GAWD | **100.0** | **92.2** | **45.5** | **83.8** | 93.8 | **90.3** | 760s |

(a) UCI Message Dataset

| Method | Precision @10 | Precision @20 | Recall @10 | Recall @20 | AUC | AP | Time |
|---|---|---|---|---|---|---|---|
| node2vec | 10.0 | 5.0 | 4.0 | 4.0 | 58.7 | 4.6 | 24.6s |
| graph2vec | 10.0 | 5.0 | 4.0 | 4.0 | 59.0 | 6.1 | 1.1s |
| Noble *et al.* | 0.0 | 0.0 | 0.0 | 0.0 | 51.7 | 3.2 | 7768s |
| Subdue-W | 10.0 | 5.0 | 4.0 | 4.0 | 48.2 | 4.9 | 8443s |
| GAWD | **90.0** | **85.0** | **36.0** | **68.0** | **82.1** | **73.8** | 207s |

(b) Enron Email Dataset

| Method | Precision @200 | Precision @400 | Recall @200 | Recall @400 | AUC | AP | Time |
|---|---|---|---|---|---|---|---|
| node2vec | 5.0 | 3.0 | 2.1 | 2.5 | 47.0 | 30. | 31.3s |
| graph2vec | 3.5 | 4.5 | 1.5 | 3.8 | 54.0 | 3.7 | 12.7s |
| Noble *et al.* | 3.0 | 3.1 | 1.2 | 2.6 | 50.6 | 3.1 | 460s |
| Subdue-W | 82.0 | 74.0 | 34.2 | 61.7 | 76.0 | 59.8 | 477s |
| GAWD | **100.0** | **81.5** | **41.7** | **67.9** | **88.0** | **71.0** | 75s |

(c) Accounting Dataset

| Method | Precision @200 | Precision @400 | Recall @200 | Recall @400 | AUC | AP | Time |
|---|---|---|---|---|---|---|---|
| node2vec | 2.5 | 3.3 | 1.0 | 0.8 | 48.3 | 2.8 | 25.9s |
| graph2vec | 0.0 | 2.0 | 1.7 | 3.1 | 49.6 | 3 | 14s |
| Noble *et al.* | 3.0 | 3.0 | 1.3 | 2.5 | 50.0 | 3 | 1426s |
| Subdue-W | 63.5 | 35 | 26.6 | 29.3 | 71.5 | 36.8 | 6584s |
| GAWD | **100.0** | **89.0** | **41.8** | **74.5** | **95.3** | **90.2** | 176s |

(d) Random Accounting Dataset

Fig. 3: **GAWD significantly outperforms all the baselines:** We show the performance of GAWD and structure-based and embedding-based baselines on four real-world and random graph datasets.

TABLE III: Summary of graph databases

| Name | Graphs | Nodes [min, max] | Edges [min, max] |
|---|---|---|---|
| UCI Message Dataset [27] | 3320 | [2, 159] | [1, 193] |
| Enron Email Dataset [29] | 843 | [2, 87] | [1, 127] |
| Accounting Dataset | 16,026 | [2, 13] | [1, 20] |
| Random Accounting Dataset | 15,935 | [2, 13] | [1, 18] |

*2) Evaluation Metric:* Datasets may originally contain anomalies, but we do not have ground truth. As shown in Figure 4, there originally exist multiple graphs with high anomaly scores along the horizontal axis, which strongly disturb the quality of evaluation. To solve this, rather than comparing all graphs in absolute terms, we look at the relative change in anomaly scores before and after injection. We quantify the relative change as $RC_i = \frac{a_i^{\text{injected}} - a_i^{\text{original}}}{a_i^{\text{original}}} * 100\ (\%)$.

To evaluate the performance of anomaly detection, we use precision at $k$, recall at $k$, Area Under Curve (AUC) and Average Precision (AP) as our evaluation metrics. The choices of $k$ are dependent on the database size since $k$ cannot exceed the number of injected graphs.

*3) Baselines:* We compare GAWD with 4 baselines:

- **Noble *et al.*** [5] iteratively finds the substructure generating the largest compression, and then assigns anomaly scores based on the compression rate in each iteration.
- **Subdue-W** follows [5] but additionally discretizes edge weights into labels by ten bins with equal size.
- **node2vec** [4] embeds each node into a vector, then inputs sum of node vectors in each graph to Isolation Forest [13].
- **graph2vec** [3] embeds each graph into a vector and inputs it to Isolation Forest [13].

### B. Effectiveness

In Table 3, GAWD outperforms most of the baselines significantly on all the datasets. Noble *et al.* and graph2vec fail as it cannot handle edge weights. GAWD shows 31.6%, 1365%, 18.7% and 145% improvement over Subdue-W in average precision on four datasets respectively, highlighting the insufficiency of discretization to handle edge weights. Even if node2vec accepts edge weights, it is not sensitive enough to detect the anomalies on weights.

The effectiveness of GAWD shows the necessity of handling numerical values instead of discretizing them into labels. In addition to improving performance on anomaly detection, we simultaneously maintain interpretability, where the common substructures identified in the course of iteratively compressing the database provide a peek into the expected structural patterns in the database, and the lack thereof in anomalous graphs.

### C. Scalability

To quantify the scalability, we empirically vary the (i) number of total edges as well as (ii) number of graphs in the database, both of which highly correlate with running time. As shown in Table III, the total number of graphs in the first experiment is 12,617, and the total edge number in the second experiment is 24,137. As shown in Figure 5, GAWD scales linearly w.r.t. both variables, with $R^2$ scores of linear-fit models higher than 0.97.

As shown in Figure 1, GAWD achieves the best trade-off between detection performance and running time comparing to the state-of-the-art approaches. In type injection, GAWD is 3.7 times faster than Noble *et al.* and the average precision is 1.3 times higher than node2vec; in path injection, GAWD is 3.9 times faster than Noble *et al.* and the average precision is also 1.1 times higher; in weight injection, GAWD is 4.1 times faster than Subdue-W and the average precision is also 1.5 times higher.

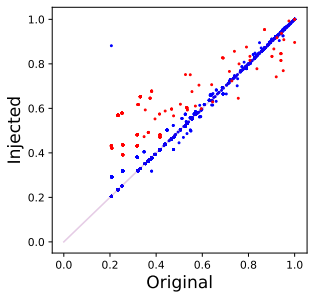



Fig. 4: **Anomaly score before (original) vs. after (injection) of each graph:** Database originally contains many graphs with high scores. Red dots depict the graphs being injected.

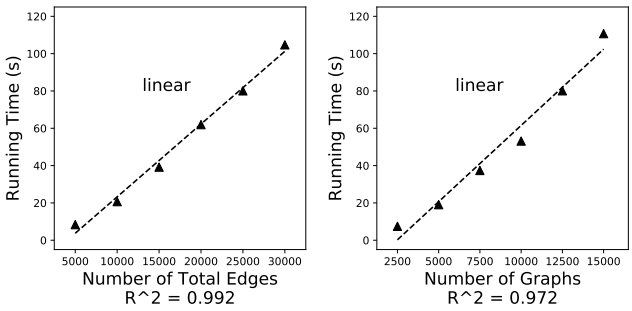



Fig. 5: **GAWD is scalable:** linear on the number of total edges (left) and the number of graphs (right).

## VI. Conclusion

We present GAWD, addressing the graph anomaly detection problem in a directed weighted graph database. Using an MDL-based approach for encoding, GAWD iteratively identifies the "best" substructure yielding the largest compression of the database. Our novel encoding scheme includes lossless encoding as well as ability to handle weighted graphs. Experiments on four datasets with injected anomalies shows GAWD achieves superior results among state-of-the-art baselines.

## References


[1] D. J. Cook and L. B. Holder, "Substructure discovery using minimum description length and background knowledge," *Journal of Artificial Intelligence Research*, vol. 1, pp. 231–255, 1993.

[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[3] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," *arXiv preprint arXiv:1707.05005*, 2017.

[4] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the ACM SIGKDD*, 2016, pp. 855–864.

[5] C. C. Noble and D. J. Cook, "Graph-based anomaly detection," in *Proceedings of the ACM SIGKDD*, 2003, pp. 631–636.

[6] W. Eberle and L. Holder, "Discovering structural anomalies in graph-based data," in *Proceedings of the IEEE International Conference on Data Mining Workshops*. IEEE, 2007, pp. 393–398.

[7] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *Proceedings of the ACM SIGKDD*, 2016, pp. 1035–1044.

[8] R. A. Bridges, J. D. Jamieson, and J. W. Reed, "Setting the threshold for high throughput detectors: A mathematical approach for ensembles of dynamic, heterogeneous, probabilistic anomaly detectors," in *Proceedings of the IEEE BigData*. IEEE, 2017, pp. 1071–1078.

[9] N. Stojanovic, M. Dinic, and L. Stojanovic, "A data-driven approach for multivariate contextualized anomaly detection: Industry use case," in *Proceedings of the IEEE BigData*. IEEE, 2017, pp. 1560–1569.

[10] Z. Chen, X. Yu, Y. Ling, B. Song, W. Quan, X. Hu, and E. Yan, "Correlated anomaly detection from large streaming data," in *Proceedings of the IEEE BigData*. IEEE, 2018, pp. 982–992.

[11] M.-C. Lee, Y. Zhao, A. Wang, P. J. Liang, L. Akoglu, V. S. Tseng, and C. Faloutsos, "AutoAudit: mining accounting and time-evolving graphs," in *Proceedings of the IEEE BigData*. IEEE, 2020.

[12] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the ACM SIGMOD*, 2000, pp. 93–104.

[13] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proceedings of the IEEE ICDM*. IEEE, 2008, pp. 413–422.

[14] T. Pevnỳ, "Loda: Lightweight on-line detector of anomalies," *Machine Learning*, vol. 102, no. 2, pp. 275–304, 2016.

[15] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *Proceedings of the ACM SIGMOD*, 2000, pp. 427–438.

[16] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang, "A novel anomaly detection scheme based on principal component classifier," Miami Univ Coral Gables Fl Dept of Electrical and Computer Engineering, Tech. Rep., 2003.

[17] Y. Zhao, Z. Nasrullah, and Z. Li, "Pyod: A python toolbox for scalable outlier detection," *Journal of Machine Learning Research*, vol. 20, no. 96, pp. 1–7, 2019. [Online]. Available: http://jmlr.org/papers/v20/19-011.html

[18] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[19] L. Akoglu, M. McGlohon, and C. Faloutsos, "Oddball: Spotting anomalies in weighted graphs," in *Proceedings of PAKDD*. Springer, 2010, pp. 410–421.

[20] M. Davis, W. Liu, P. Miller, and G. Redpath, "Detecting anomalies in graphs with numeric labels," in *Proceedings of the ACM CIKM*, 2011, pp. 1197–1202.

[21] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data mining and knowledge discovery*, vol. 29, no. 3, pp. 626–688, 2015.

[22] K. Henderson, B. Gallagher, L. Li, L. Akoglu, T. Eliassi-Rad, H. Tong, and C. Faloutsos, "It's who you know: graph mining using recursive structural features," in *Proceedings of the ACM SIGKDD*, 2011, pp. 663–671.

[23] U. Kang, J.-Y. Lee, D. Koutra, and C. Faloutsos, "Net-ray: visualizing and mining billion-scale graphs," in *Proceedings of the PAKDD*. Springer, 2014, pp. 348–361.

[24] N. Gupta, D. Eswaran, N. Shah, L. Akoglu, and C. Faloutsos, "Beyond outlier detection: Lookout for pictorial explanation," in *Proceedings of the ECML*. Springer, 2018, pp. 122–138.

[25] E. K. Chong and S. H. Zak, *An introduction to optimization*. John Wiley & Sons, 2004.

[26] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Proceedings of IEEE ICDM*. IEEE, 2002, pp. 721–724.

[27] T. Opsahl and P. Panzarasa, "Clustering in weighted networks," *Social networks*, vol. 31, no. 2, pp. 155–163, 2009.

[28] N. K. Ahmed, R. Rossi, J. B. Lee, T. L. Willke, R. Zhou, X. Kong, and H. Eldardiry, "Learning role-based graph embeddings," *arXiv preprint arXiv:1802.02896*, 2018.

[29] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *Proceedings of European Conference on Machine Learning*. Springer, 2004, pp. 217–226.

[30] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *Proceedings of the ACM SIGKDD*, 2018, pp. 2672–2681.

[31] L. Zheng, Z. Li, J. Li, Z. Li, and J. Gao, "Addgraph: Anomaly detection in dynamic graph using attention-based temporal gcn," in *Proceedings of the IJCAI*, 2019, pp. 4419–4425.

[32] D. J. Kleitman and D.-L. Wang, "Algorithms for constructing graphs and digraphs with given valences and factors," *Discrete Mathematics*, vol. 6, no. 1, pp. 79–88, 1973.

## APPENDIX

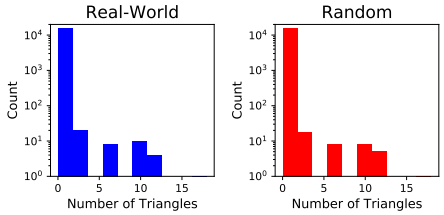
### A. Random Transaction Graph Database Generation

*1) Algorithm:* Algorithm 3 illustrates the procedure of generation. In line 2, we extract the statistics from the given transaction graph database $G$, where $D_{in}$ and $D_{out}$ denote the in- and out-degree sequences for each graph respectively, $P_t$ denotes the probability distribution of node types in the database, $P_e$ denotes the probability distribution of edge types, and $P_w$ denotes the probability distribution of weights given an edge type. We then create the random graph by in- and out-degree sequences of each graph with Directed Havel Hakimi Graph [32] in line 4. In line 5, we randomly initialize the first node and assign the label by $P_t$. We then greedily assign labels to other nodes by $P_e$ in line 6-11. To those unassigned nodes and edges, we finish the assignment after the greedy one.

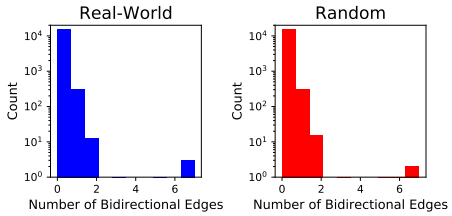
**Data:** A transaction graph database $G$
**Result:** A random transaction graph database $G^*$
1 $G^* \leftarrow \emptyset$;
2 Extract $D_{in}, D_{out}, P_t, P_e$ and $P_w$ from $G$;
3 **for** $d_{in} \in D_{in}, d_{out} \in D_{out}$ **do**
4  Create a random graph $g$ by [32] with $d_{in}$ and $d_{out}$;
5  Randomly select a node $n_i$ and assign a label by $P_t$;
6  **while** $n_i$ *exists any neighbour without label* **do**
7  Randomly select $n_j$ from neighbours without label;
8  Assign a label to $n_j$ by $P_e$;
9  Assign a weight to edge $(n_i, n_j)$ by $P_w$;
10  $n_i \leftarrow n_j$;
11  **end**
12  Assign labels to those unlabeled nodes by $P_t$;
13  Assign weights to those unweighted edges by $P_w$;
14  $G^* \leftarrow G^* \cup g$;
15 **end**
16 Return $G^*$;

**Algorithm 3:** Random Transaction Graph Database Generator

*2) Experimental Results:* We examine the similarity between the real-world and random transaction graph database. Since we use the same degree sequences to generate the graphs, the distribution of node and edge number are the same as well. Figure 6 illustrates the similar distributions of specific structures in the real-world and random transaction graphs. Number of triangles is a common statistical measure for the graphs, and bidirectional edges represent an essential meaning in transaction graphs, denoting the mutual money transfers. In Figure 7, we show that the distributions node type, edge weight and edge type are extremely similar, except for some of the ones with small number of occurrences. Figure 8 depicts the edge weight distribution for each edge type, where we can find that they are also very similar. In summary, experimental results demonstrate that our random transaction graph database statistically follows the real-world one.

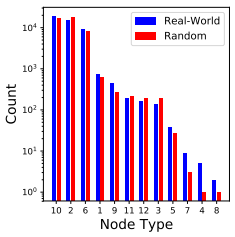



(a) Triangle Number

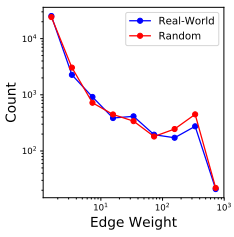



(b) Bidirectional Edge Number

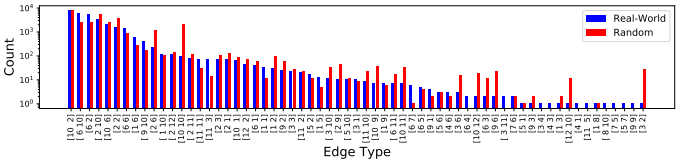
Fig. 6: Comparison on distributions of triangle number and bidirectional edge number




(a) Node Type

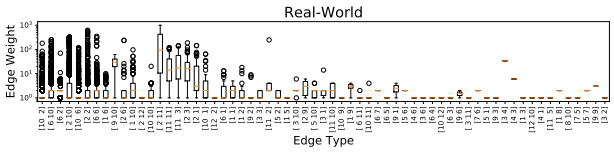



(b) Edge Weight

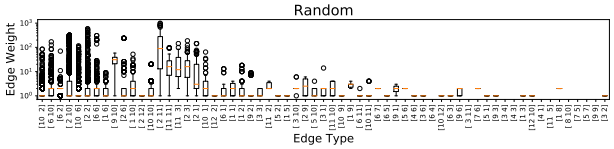



(c) Edge Type

Fig. 7: Comparison on distributions of node type, edge weight and edge type




(a) Real-World Database




(b) Random Database

Fig. 8: Box plot of edge type versus edge weight