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 among 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.

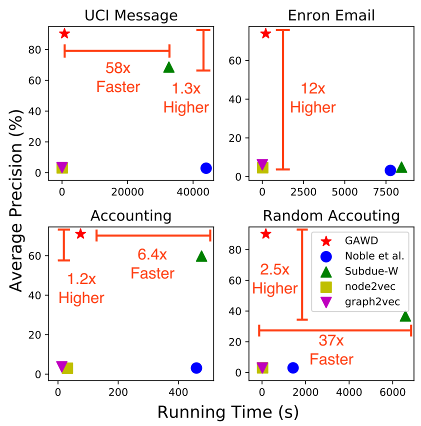

Fig. 1: **GAWD wins on both effectiveness and scalability:** We evaluate four datasets and show the big gap between it and competitors w.r.t. average precision and run time.

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.

Experimental results on four datasets with injected anomalies show that GAWD outperforms both graph embedding- and structure-based methods.

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.

Property \ Method	node2vec [4]	graph2vec [3]	Noble *et al.* [5]	GBAD [6]	OddBall [8]	Yagada [9]	GAWD
Handle Graph Database	✓	✓	✓	✓		✓	✓
Handle Node Labels		✓	✓	✓	✓	✓	✓
Handle Edge Weights	✓				✓	✓*	✓
Lossless							✓
Scale Linearly	✓	✓			✓		✓

II. Related Work

Graph-based anomaly detection has been studied for its applicability to real-world scenarios. Careful scrutiny of these studies can be found in [7]. 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. OddBall [8] detects anomalous vertices in a *single* weighted graph, but does not extend to graph databases. For numerical weights, Yagada [9] uses discretization to assign edges with discrete labels. However, discretization loses information and underperforms as we demonstrate in our experiments. Graph embedding can be used to detect anomalous graphs in conjunction with off-the-shelf anomaly detectors such as Isolation Forest [10]. 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 contrasts GAWD against the existing state-of-the-art.

III. Problem Definition

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

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 the 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.

```
  Data: A graph database
  Result: Anomaly scores for all graphs
1 while True do
2     Detect frequent substructures in graph database;
3     if No substructure is found then
4         Break;
5     end
6     Identify substructure that best compresses database;
7     Compress graphs by this substructure;
8 end
9 Compute anomaly scores by compression rate;
```

Algorithm 1: General Framework for Graph Anomaly Detection (Blue text emphasizes the changes in GAWD)

In particular, the existing method in [5] detects the frequent substructures in Line 2 by beam search. To identify the best substructure, they minimize the total description length of graphs in the database in Line 6. However, this method only takes the compressed node and edge information into consideration (*lossy encoding*). They then compress the graphs by that substructure in Line 7. This process iterates until no more substructure is found. Different from [5], GAWD replaces the beam search in Line 2 by *gSpan* [11], which is a much faster subgraph mining technique and preserves substructure quality; we design a novel graph encoding scheme, used in Line 6, which accepts edge weight (described in Subsection IV-A) and is lossless (described in Subsection 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 (by *gSpan*), 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:

$$w^*_{P_j} = \min_w \sum_{(s,t)\in E_{j,(u,v)}} L(w, w_{P_j}(s,t)) \tag{1}$$

While not convex, the optimization in (1) is only 1-dimensional and hence relatively easy to solve. We employ the Dichotomous Search algorithm [12], which returns the optimal solution in most cases. The search is also efficient, taking $O(|E_{j,(u,v)}| \log_2 R)$, 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 instead of discretizing edge weights into labels, our encoding scheme handles the numeric value and is lossless.

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.

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

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

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:

$$L(b_y) = \log_2(|V_j|) \text{ bits} . \tag{3}$$

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.

C. Overall Algorithm

Algorithm 2 gives the steps of GAWD. We use *gSpan* [11] 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:

```
Data: A database $\mathcal{G} = \{G_1, ..., G_I\}$, min support range
      $(ms_{\max}, ms_{\min})$, decay rate $d$ (i.e., 0.9 be 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      $\mathcal{P}_j = gSpan(\mathcal{G}, ms)$;
4      Discover $w^*_{P_j}(u,v)$ for all $P_j \in \mathcal{P}_j$ (See § IV-A1);
5      Identify $P^*_j \in \mathcal{P}_j$ yielding largest (positive) compression;
6      if no $P^*_j$ is found then
7          $ms := ms * d$;
8          Continue;
9      end
10     Compress $\mathcal{G}$ by $P^*_j$ and save $c^j_i$ in (4) for all $G_i \in \mathcal{G}$;
11     $j := j + 1$;
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

$$c^j_i = \frac{DL_{j-1}(G_i) - DL_j(G_i)}{DL_0(G)}, \tag{4}$$

where $DL_j(G_i)$ is the description length of G_i after j iterations. Finally, we compute the anomaly scores in Line 13.

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.*

The proof is in the full version at https://bit.ly/3mgjRdn.

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 source code and datasets along with detailed descriptions are shared at https://github.com/mengchillee/GRANDDIWE. Experiments are run on a 3.2 GHz CPU on a Linux server.

A. Settings

1) Datasets: We use four datasets illustrated in Table III. The Random Accounting Dataset we generated statistically follows the original graphs (see full version for more details). 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 [7] suggested, which had also been done by several other studies [13], [14]. 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

TABLE II: **Performance of GAWD and baselines on the four datasets: GAWD significantly outperforms both structure-based and embedding-based baselines in all real-world and random graph datasets.**

Method	UCI Message Dataset				Enron Email Dataset				Accounting Dataset				Random Accounting Dataset			
	Precision		AUC	AP	Precision		AUC	AP	Precision		AUC	AP	Precision		AUC	AP
	@45	@90			@10	@20			@200	@400			@200	@400		
node2vec	6.7	5.6	47.6	3.1	10.0	5.0	58.7	4.6	5.0	3.0	47.0	3.0	2.5	3.3	48.3	2.8
graph2vec	2.2	1.1	48.7	3.1	10.0	5.0	59.0	6.1	3.5	4.5	54.0	3.7	0.0	2.0	49.6	3.0
Noble *et al.*	0.0	0.0	47.6	3.1	0.0	0.0	51.7	3.2	3.0	3.1	50.6	3.1	3.0	3.0	50.0	3.0
Subdue-W	**100.0**	60.0	**94.8**	68.6	10.0	5.0	48.2	4.9	82.0	74.0	76.0	59.8	63.5	35.0	71.5	36.8
GAWD	**100.0**	**92.2**	93.8	**90.3**	**100.0**	**85.0**	**79.1**	**71.8**	**100.0**	**81.5**	**88.0**	**71.0**	**100.0**	**89.0**	**95.3**	**90.2**

TABLE III: Summary of graph databases

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

the boxplot of weights for $(t(u), t(v))$ edges. This aims to simulate unusual behaviors between the users or accounts.

2) Evaluation Metric: Our dataset may originally contain anomalies, but we do not have ground truth. There originally exist multiple graphs with high anomaly scores 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 $RC_i = \frac{a_i^{\text{injected}} - a_i^{\text{original}}}{a_i^{\text{original}}} * 100$ (%).

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 Noble *et al.* but additionally discretizes edge weights into labels.
- **node2vec** [4] embeds each node into a vector, then inputs sum of node vectors in each graph to Isolation Forest [10].
- **graph2vec** [3] embeds each graph into a vector and inputs it to Isolation Forest [10].

B. Effectiveness

As shown in Table II, GAWD outperforms most of the baselines significantly on all the datasets. Recall and run time are also reported in the full version. 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, which highlights the insufficiency of discretization to handle edge weights. Although node2vec accepts edge weights, it is not sensitive enough to detect the anomalies on weights.

C. Scalability

We empirically vary the (i) number of total edges as well as (ii) number of graphs in the database. The total number of graphs in the first experiment is about 12K, and the total edge number in the second one is 24K. In Figure 2, GAWD scales linearly w.r.t. both variables, with R^2 scores of linear-fit models higher than 0.97. In Figure 1, we further shown that structure-based baselines are not scalable; Though embedding-based baselines scale linearly, they fail to detect the anomalies.

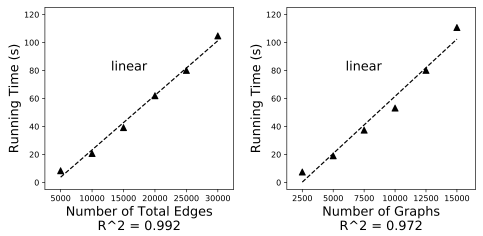

Fig. 2: **GAWD is scalable**: linear on the number of total edges (left) and the number of graphs (right).

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 provides lossless encoding, and also has the ability to handle graphs with numeric edge weights. We presented experiments on four datasets with injected anomalies, where 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," *Jour. of AI Research*, vol. 1, pp. 231–255, 1993.

[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 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 *ACM SIGKDD*, 2016, pp. 855–864.

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

[6] W. Eberle and L. Holder, "Discovering structural anomalies in graph-based data," in *ICDMW*. IEEE, 2007, pp. 393–398.

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

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

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

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

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

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

[13] 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 *ACM SIGKDD*, 2018, pp. 2672–2681.

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

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

[16] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *ECML*. Springer, 2004, pp. 217–226.

VII. Appendix

Hi