Detecting Anomalous Graphs in Labeled Multi-Graph Databases

HUNG T. NGUYEN, Princeton University, USA
PIERRE J. LIANG, Carnegie Mellon University, USA
LEMAN AKOGLU, Carnegie Mellon University, USA

Within a large database $\mathcal{G}$ containing graphs with labeled nodes and directed, multi-edges; how can we detect the anomalous graphs? Most existing work are designed for plain (unlabeled) and/or simple (unweighted) graphs. We introduce CODETECT, the *first* approach that addresses the anomaly detection task for graph databases with such complex nature. To this end, it identifies a small representative set $\mathcal{S}$ of structural patterns (i.e., node-labeled network motifs) that losslessly compress database $\mathcal{G}$ as concisely as possible. Graphs that do not compress well are flagged as anomalous. CODETECT exhibits two novel building blocks: (*i*) a motif-based lossless graph encoding scheme, and (*ii*) fast memory-efficient search algorithms for $\mathcal{S}$. We show the effectiveness of CODETECT on transaction graph databases from three different corporations and statistically similar synthetic datasets, where existing baselines adjusted for the task fall behind significantly, across different types of anomalies and performance metrics.

CCS Concepts: • **Information systems → Data mining**; • **Computing methodologies → Anomaly detection**.

Additional Key Words and Phrases: Graph anomaly detection, Graph encoding, Graph motifs

1 INTRODUCTION

Given hundreds of thousands of annual transaction records of a corporation, how can we identify the abnormal ones, which may indicate entry errors or employee misconduct? How can we spot anomalous daily email/call interactions or software programs with bugs?

We introduce a novel anomaly detection technique called CODETECT, for *node-Labeled, Directed, Multi-graph (LDM) databases* which emerge from many applications. Our motivating domain is accounting, where each transaction record is represented by a graph in which the nodes are accounts and directed edges reflect transactions. Account types (revenue, expense, etc.) are depicted by (discrete) labels and separate transactions between the same pair of accounts create edge multiplicities. The problem is then identifying the anomalous graphs within LDM graph databases. In these abstract terms, CODETECT applies more broadly to other domains exhibiting graph data with such complex nature, e.g., detecting anomalous employee email graphs with job titles as labels, call graphs with geo-tags as labels, control flow graphs with function-calls as labels, etc. What is more, CODETECT can handle simpler settings with any subset of the LDM properties.

Graph anomaly detection has been studied under various non-LDM settings. Most of these work focus on detecting anomalies within a *single* graph; either plain (i.e., unlabeled), attributed (nodes exhibiting an array of (often continuous) features), or dynamic (as the graph changes over time) [1, 2, 14, 20, 28, 38, 39, 46] (See Table 1 for overview). None of these applies to our setting, as we are to detect graph-level anomalies within a

Authors' addresses: Hung T. Nguyen, hn4@princeton.edu, Princeton University, Princeton, NJ, USA; Pierre J. Liang, Carnegie Mellon University, Pittsburgh, PA, USA, liangj@andrew.cmu.edu; Leman Akoglu, Carnegie Mellon University, Pittsburgh, PA, USA, lakoglu@andrew.cmu.edu.

1556-4681/2022/5-ART $15.00
https://doi.org/10.1145/3533770

graph *database*. There exist related work for node-labeled graph databases [34], which however does not handle multi-edges, and as we show in the experiments (Sec. 6), cannot tackle the problem well.

Recently, general-purpose embedding/representation learning techniques achieve state-of-the-art results in graph classification tasks [15, 16, 19, 31, 33, 37]. However, they do not tackle the anomaly detection problem *per se*—the embeddings need to be fed to an off-the-shelf vector outlier detector. Moreover, most embedding methods [15, 16, 19] produce *node* embeddings; how to use those for graph-level anomalies is unclear. Trivially aggregating node representations, e.g., by mean or max pooling, to obtain the entire-graph representation provides suboptimal results [31]. Graph embedding techniques [31, 37] as well as graph kernels [45, 54] (paired with a state-of-the-art detector), yield poor performance as we show through experiments (Sec. 6), possibly because embeddings capture general patterns, leaving rare structures out, which are critical for anomaly detection.

Our main contributions are summarized in the following:

- **Problem Formulation:** Motivated by application to business accounting, we consider the anomaly detection problem in labeled directed multi-graph (LDM) databases and propose CODEtect; (to our knowledge) the *first* method to detect anomalous graphs with such complex nature (Sec. 2). CODEtect also generally applies to simpler, non-LDM settings. The main idea is to identify a few representative network motifs that are used to encode the database in a lossless fashion as succinctly as possible. CODEtect then flags those graphs that do not compress well under this encoding as anomalous (Sec. 3).
- **New Encoding & Search Algorithms:** The graph encoding problem is two-fold: how to encode and which motifs to encode with. To this end, we introduce (1) new lossless motif and graph encoding schemes (Sec. 4), and (2) efficient search algorithms for identifying key motifs with a goal to minimize the total encoding cost (Sec. 5).
- **Real-world Application:** In collaboration with industry, we apply our proposed techniques to annual transaction records from three different corporations, from small- to large-scale. We show the superior performance of CODEtect over existing baselines in detecting injected anomalies that mimic certain known malicious schemes in accounting. Case studies on those as well as the public Enron email database further show the effectiveness of CODEtect in spotting noteworthy instances (Sec. 6). To facilitate reproducibility, we also confirm our performance advantages on statistically similar datasets resembling our real-world databases.

Reproducibility. All source code as well as public-domain and synthetic data are shared at https://bit.ly/2P0bPZQ.

2 RELATED WORK

Graph Anomaly Detection: Graph anomaly detection has been studied under various settings for plain/attributed, static/dynamic, etc. graphs, including the most recent deep learning based approaches [1, 11, 14, 20, 38, 39, 46, 57] (See [2] and [27] for a survey.) These works focus on detecting *node/edge/subgraph anomalies* within a *single* graph, none of which applies to our setting, as we are to detect anomalous graphs (or *graph-level anomalies*) within a graph *database*.

On anomalous graph detection in graph databases, Gbad [12] has been applied to flag graphs as anomalous if they experience low compression via discovered substructures over the iterations. Further, it has been used to identify graphs that contain substructures S' with small differences (a few modifications, insertions, or deletions) from the best one S, which are attributed to malicious behavior [12]. Gbad also has very high time complexity due to the nested searches for substructures to compress the graphs through many iterations (failed to complete on multiple cases of our experiments - see Section 6). Our work is on the same lines with these work in principle, however our encoding scheme is lossless. Moreover, these work cannot handle graphs with weighted/multi edges. There exist other graph anomaly detection approaches [14, 28], however none of them simultaneously handle node-labeled graphs with multi-edges. SnapSketch [36] was introduced recently as an unsupervised graph

Table 1. Comparison with popular approaches to graph anomaly detection, in terms of distinguishing properties.

Methods vs. Properties		Graph database	Node-labeled	Multi/ Weighted	Directed	Anomaly detection
Graph Emb.	node2vec[16], GraphSAGE[19]		✔	✔	✔	
	graph2vec[31], Metagraph2vec[15], GIN[53]	✔	✔		✔	
	PATCHY-SAN[33], MA-GCNN[37], Deep Graph Kernels [54]	✔	✔	✔	✔	
Graph Anom. Detect.	OddBall[1]			✔	✔	✔
	FocusCO[39], AMEN[38], DOMINANT[11]		✔		✔	✔
	GAL[57]			✔	✔	✔
	CoreScope[46]					✔
	FRAUDAR[20]				✔	✔
	StreamSpot[28], GBAD[12]	✔	✔		✔	✔
	SpotLight[14]	✔		✔	✔	✔
	SNAPSKETCH[36]	✔		✔		✔
	GLOCALKD[26]	✔	✔			✔
	CODETECT [this paper]	✔	✔	✔	✔	✔

representation approach for intrusion detection in a graph stream and showed better detection than previous works [14, 28], however, SNAPSKETCH was originally designed to work on undirected graphs. Note also that these works [14, 28, 36] focus on graph streams, i.e., time-ordered graphs, and may not work well in our setting of *unordered* graph databases. We present a qualitative comparison of related work to CODETECT in Table 1.

Graph Embedding for Anomaly Detection: Recent graph embedding methods [11, 15, 16, 19, 31, 33, 37, 57] and graph kernels [45, 54] find a latent representation of node, subgraph, or the entire graph and have been shown to perform well on classification and link prediction tasks. However, graph embedding approaches, like [15, 16, 19], learn a node representation, which is difficult to use directly for detecting anomalous graphs. Peng et al. [37] propose a graph convolutional neural network via motif-based attention, but, this is a supervised method and, thus, not suitable for anomaly detection. Our experimental results show that other recent graph embedding [31] and graph kernel methods [54], that produce a direct graph representation, when combined with a state-of-the-art anomaly detector have low performance and are far less accurate than CODETECT. Concurrent to our work, graph neural networks for anomalous graph detection is studied in [26, 56] that examines end-to-end graph anomaly detection. Further, [6] investigates outlier resistant architectures for graph embedding. A key challenge, in general, for deep learning based models for *unsupervised* anomaly detection is their sensitivity to many hyper-parameter settings (including those for regularization: such as weight decay, and drop-out rate; optimization: such as learning rate, and achitecture: such as depth, width, etc.), which are not straightforward to set in the absence of any ground-truth labels. Distinctly, our work leverages the Minimum Description Length principle and does not exhibit any hyper-parameters.

Graph Motifs: Network motifs have proven useful in understanding the functional units and organization of complex systems [7, 30, 51]. Motifs have also been used as features for network classification [29], community detection [5, 55], and in graph kernels for graph comparison [45]. On the algorithmic side, several works have

designed fast techniques for identifying significant motifs [8, 9, 21, 22], where a sub-graph is regarded as a motif only if its frequency is higher than expected under a network null model.

Prior works on network motifs mainly focus on 3- or 4-node motifs in undirected unlabeled/plain graphs [4, 13, 42, 50], either using subgraph frequencies in the analysis of complex networks or most often developing fast algorithms for counting (e.g., triangles) (See [**?**] for a recent survey). Others have also studied directed [7] and temporal motifs [24, 35]. Most relatedly, there is a recent work on node-labeled subgraphs referred to as heterogeneous network motifs [41], where again, the focus is on scalable counting. Our work differs in using heterogeneous motifs as building blocks of a graph encoding scheme, toward the goal of anomaly detection.

Data Compression via MDL-Encoding: The MDL principle by Rissanen [40] states that the best theory to describe a data is the one that minimizes the sum of the size of the theory, and the size of the description of data using the theory. The use of MDL has a long history in itemset mining [48, 52], for transaction (tabular) data, also applied to anomaly detection [3, 47].

MDL has also been used for graph compression. Given a pre-specified list of well-defined structures (star, clique, etc.), it is employed to find a succinct description of a graph in those "vocabulary" terms [23]. This vocabulary is later extended for dynamic graphs [44]. A graph is also compressed hierarchically, by sequentially aggregating sets of nodes into super-nodes, where the best summary and associated corrections are found with the help of MDL [32].

There exists some work on attributed graph compression [49], but the goal is to find super-nodes that represent a set of nodes that are homogeneous in some (user-specified) attributes. SUBDUE [34] is one of the earliest work to employ MDL for substructure discovery in node-labeled graphs. The aim is to extract the "best" substructure S whose encoding plus the encoding of a graph after replacing each instance of S with a (super-)node is as small as possible.

3 PRELIMINARIES & THE PROBLEM

As input, a large set of J graphs $\mathcal{G} = \{G_1, \ldots, G_J\}$ is given. Each graph $G_j = (V_j, E_j, \tau)$ is a directed, node-labeled, multi-graph which may contain multiple edges that have the same end nodes. $\tau : V_j \rightarrow \mathcal{T}$ is a function that assigns labels from an alphabet $\mathcal{T}$ to nodes in each graph. The number of realizations of an edge $(u, v) \in E_j$ is called its *multiplicity*, denoted $m(u, v)$. (See Fig. 1(a) for example.)

Our motivating domain is business accounting, in which each G_j corresponds to a graph representation of what-is-called a "journal entry": a detailed transaction record. Nodes capture the unique accounts associated with the record, *directed* edges the transactions between these accounts, and node *labels* the financial statement (FS) account types (e.g., assets, liabilities, revenue, etc.). Bookkeeping data is kept as a chronological listing (called General Ledger) of each separate business transaction, where multiple transactions involving same account-pairs generate multi-edges between two nodes.

Our high-level idea for finding anomalous graphs in database $\mathcal{G}$ is to identify key characteristic *patterns* of the data that "explain" or compress the data well, and flag those graphs that do not exhibit such patterns as expected—simply put, graphs that do not compress well are anomalous. More specifically, graph patterns are substructures or subgraphs, called *motifs*, which occur frequently within the input graphs. "Explaining" the data is encoding each graph using the frequent motifs that it contains. The more frequent motifs we use for encoding, the more we can compress the data; simply by encoding the existence of each such motif with a short code.

The goal is to find a (small) set of motifs that compresses the data the best. Building on the Minimum Description Length (MDL) principle [17], we aim to find a model, namely a *motif table* (denoted MT) that contains a carefully selected subset of graph motifs, such that the total code length of (1) the model itself plus (2) the encoding of the data using the model is as small as possible. In other words, we are after a small model that compresses the data the most. The two-part objective of minimizing the total code length is given as follows.

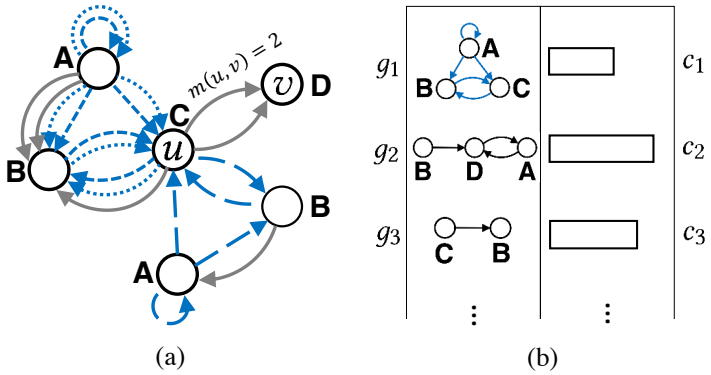


Fig. 1. (a) E.g. node-labeled multi-graph where capital letters denote the node labels, and $m(u, v)$ depicts multiplicity of edge (u, v) ; (b) Example motif table; 1st col. lists the motifs, and 2nd col. provides the corresponding codes, bar width depicting code length. The blue edges in (a) sharing the same dash type depict three different occurrences of the top motif (i.e. g_1) in the motif table in (b).

$$\underset{MT \subseteq \mathcal{MT}}{\text{minimize}} \quad L(MT, \mathcal{G}) = \underbrace{L(MT)}_{\text{model code length}} + \underbrace{L(\mathcal{G}|MT)}_{\text{data code length}} , \tag{1}$$

where $\mathcal{MT}$ denotes the set of all possible candidate motif tables. The first term can be seen as a model regularizer that penalizes using an unneccesarily large set of motifs to explain the data. The second term is the compression length of the data with the (selected) motifs and decomposes as $L(\mathcal{G}|MT) = \sum_j L(G_j|MT)$ since individual journals are independent. The encoding length $L(G_j|MT)$ is also the *anomaly score* for the jth graph—the larger, the more anomalous.

As such, we have a combinatorial subset selection problem toward optimizing Eq. (1). To this end, we address two subproblems outlined below.

Problem 1. *Our graph encoding problem is two-fold: (1) how to encode, and (2) which motifs to encode with, or,*

(1) ***Encoding Schemes*** *(Sec. 4): Define schemes for (i) $L(MT)$; encoding the motifs in MT, and (ii) $L(G_j|MT)$; encoding a graph with the given motifs; and*

(2) ***Search Algorithm*** *(Sec. 5): Derive a subset selection algorithm for identifying a set of motifs to put in MT.*

4 ENCODING SCHEMES

MDL-encoding of a dataset with a model can be thought to involve a Sender and a Receiver communicating over a channel, where the Sender—who has the sole knowledge of the data—generates a bitstring based on which the Receiver can reconstuct the original data on their end *losslessly*. To this end, the Sender first sends over the model, in our case the motif table MT, which can be thought as a "code-book" that establishes a certain "language" between the Sender and the Receiver. The Sender then encodes the data instances using the code-words in the "code-book", in our case the graph G_j's using the motifs in the MT.

It is important to note that existing works [9, 10] presented different graph/substructure encoding schemes, however, each has its own limitation and does not work in our settings. The encoding in [10] is lossy and may not reflect accurate code length, while the work in [9] is restricted on simple graphs with non-overlapping nodes of

Algorithm 1 Motif Encoding

Input: Motif $g_i = (V_i, E_i)$
Output: Encoding of g_i ▶ Note: the values after symbol ▷ summed over the course of the algorithm provides the total encoding length $L(g_i)$

Encode $n_i = |V_i|$; # of nodes in g_i ▷ $L_{\mathbb{N}}(n_i)$[2]
repeat
  Pick an unmarked node $v \in V_i$ at random where $indeg(v) = 0$ (if none, pick any unmarked node), and mark v
  procedure RecurseNode($mid(v)$)
    Encode $mid(v)$; v's motif-node-ID ▷ $\log_2(n_i)$
    Encode v's node label ▷ $\log_2(T)$
    Encode # of v's out-neighbors $\mathcal{N}_{\text{out}}(v)$ ▷ $L_{\mathbb{N}}(\text{outdeg}(v))$
    Encode motif-node-IDs of $\mathcal{N}_{\text{out}}(v)$ ▷ $\log_2 \binom{n_i}{\text{outdeg}(v)}$
    for each unmarked node $u \in \mathcal{N}_{\text{out}}(v)$ **do**
      Mark u
      RecurseNode($mid(u)$)
until all nodes in V_i are marked

motifs' occurrences. Our encoding algorithm is both lossless and applicable on multi-graph with node-overlapping occurrences of motifs.

4.1 Encoding the Motif Table

The motif table MT is simply a two-column translation table that has motifs in the first column, and a unique code-word corresponding to each motif in the second column, as illustrated in Fig. 1 (b). We use $\mathcal{M}$ to refer to the set of motifs in MT. A motif, denoted by (lower-case) g, is a connected, directed, node-labeled, *simple* graph, with possible self-loops on the nodes. For $g \in \mathcal{M}$, $code_{MT}(g)$ (or c for short) denotes its code-word.[1]

To encode the motif table, the Sender encodes each individual motif g_i in MT and also sends over the code-word c_i that corresponds to g_i. Afterwards, for encoding a graph with the MT, every motif that finds an occurrence in the graph is simply communicated through its unique code-word only.

The specific materialization of the code-words (i.e., the bitstrings themselves) are not as important to us as their *lengths*, which affect the total graph encoding length. Each code length $L(c_i)$ depends on the number of times that g_i is *used* in the encoding of the graphs in $\mathcal{G}$, denoted $usage_{\mathcal{G}}(g_i)$—intuitively, the more frequently a motif is used, the shorter its code-word is; so as to achieve compression (analogous to compressing text by assigning frequent words a short code-word). Formally, the optimal prefix code length for g_i can be calculated through the Shannon entropy [40]:

$$L(c_i) \;=\; |code_{MT}(g_i)| \;=\; -\log_2[P(g_i|\mathcal{G})]\;, \tag{2}$$

where P is a probability distribution of $g_k \in \mathcal{M}$ for $\mathcal{G}$:

$$P(g_i|\mathcal{G}) = \frac{\sum_{G_j \in \mathcal{G}} usage_{G_j}(g_i)}{\sum_{g_k \in \mathcal{M}} \sum_{G_j \in \mathcal{G}} usage_{G_j}(g_k)}\;. \tag{3}$$

[1]To ensure *unique* decoding, we assume *prefix code(word)s*, in which no code is the prefix of another.

[2]MDL-optimal cost of integer k is $L_{\mathbb{N}}(k) = \log^{\star} k + \log_2 c$; $c \approx 2.865$; $\log^{\star} k = \log_2 k + \log_2(\log_2 k) + \ldots$ summing only the positive terms [40].

We provide the details of how the motif usages are calculated in the next section, when we introduce graph encoding.

Next, we present how a motif g_i is encoded. Let n_i denote the number of nodes it contains. (e.g., g_1 in Fig. 1 (b) contains 3 nodes.) The encoding progresses recursively in a DFS (Depth First Search)-like fashion, as given in Algo. 1. As noted, the encoding lengths summed over the course of the algorithm provides $L(g_i)$, which can be explicitly written as follows:

$$\begin{aligned} L(g_i) =& L_{\mathbb{N}}(n_i) + \sum_{v \in V_i} \log_2(n_i) + \log_2(T) \\ &+ L_{\mathbb{N}}(\text{outdeg}(v)) + \log_2 \binom{n_i}{\text{outdeg}(v)} \end{aligned} \tag{4}$$

Overall, the total model encoding length is given as

$$L(MT) = L_{\mathbb{N}}(T) + \sum_{g_i \in MT} \left[L(g_i) + L(c_i) \right] , \tag{5}$$

where we first encode the number of unique node labels, followed by the entries (motifs and codes) in the motif table.

4.2 Encoding a Graph given the Motif Table

To encode a given graph G_j based on a motif table MT, we "cover" its edges by a set of motifs in the MT. To formally define coverage, we first introduce a few definitions.

Definition 4.1 (Occurrence). An occurrence (a.k.a. a match) of a motif $g_i = (V_i, E_i)$ in a graph $G_j = (V_j, E_j)$ is a simple subgraph of G_j, denoted $o(G_j, g_i) \equiv g_{ij} = (V_{ij}, E_{ij})$ where $V_{ij} \subseteq V_j$ and $E_{ij} \subseteq E_j$, which is *isomorphic* to g_i, such that there exists a structure- and label-preserving bijection between node-sets V_i and V_{ij}. Graph isomorphism is denoted $g_{ij} \simeq g_i$.

Given a motif occurrence g_{ij}, we say that g_{ij} *covers* the edge set $E_{ij} \subseteq E_j$ of G_j. The task of encoding a graph G_j is then to *cover all of its edges* E_j using the motifs in MT.

Definition 4.2 (Cover Set). Given a graph G_j and a motif table MT, $\mathcal{CS}(G_j, MT)$ is a set of motif occurrences so that

- If $g_{kj} \in \mathcal{CS}(G_j, MT)$, then $g_k \in \mathcal{M}$,
- $\bigcup_{g_{kj} \in \mathcal{CS}(G_j, MT)} E_{kj} = E_j$, and
- If $g_{kj}, g_{lj} \in \mathcal{CS}(G_j, MT)$, then $E_{kj} \cap E_{lj} = \emptyset$.

We say that $\mathcal{CS}(G_j, MT)$, or $\mathcal{CS}_j$ for short, covers G_j. The last item enforces the motif occurrences to cover *non-overlapping edges* of a graph, which in turn ensures that the coverage of a graph is always unambiguous. This is mainly a computational choice—allowing overlaps would enable many possible covers, where enumerating and computing all of them would significantly increase the computational cost in the search of a motif table (See Eq. (1)).

To encode a G_j via MT, the Sender communicates the code of the motif associated with each occurrence in G_j's cover set:

$$G_j \;\rightarrow\; \{code_{MT}(g_i) \mid g_{ij} \in \mathcal{CS}(G_j, MT)\} \; .$$

However, it is not enough for the Receiver to simply know which collection of motifs occur in a graph in order to decode the graph *losslessly*. The Sender needs to encode further information regarding the *structure* of the graph. This can be achieved by, in addition to encoding *which* motif occurs, also encoding the specific

graph-node-IDs that correspond to or align with the motif-node-IDs. The motif-node-IDs of a motif g_i is simply the increasing set $\{1, \ldots, n_i\}$. Then, the sequence of matching node IDs in graph G_j is simply a permutation of n_i unique values in $\{1, \ldots, |V_j|\}$, which can be encoded using $L_{\text{perm}}(|V_j|, n_i)$ bits, where

$$L_{\text{perm}}(|V_j|, n_i) = \log_2(|V_j| \cdot |V_j - 1| \cdot \ldots \cdot |V_j - n_i + 1|) \ . \tag{6}$$

Moreover, note that a motif can occur multiple times in a graph, possibly on the same set of node IDs (due to the input graphs being *multi*-graphs) as well as on different sets. We denote the different occurrences of the same motif g_i in graph G_j's cover set by $\mathcal{O}(g_i, \mathcal{CS}_j) = \langle g_{ij}^{(1)}, g_{ij}^{(2)}, \ldots, g_{ij}^{(c_{ij})} \rangle$ where c_{ij} denotes the total count. See for example Fig. 1 where motif g_1 shown in (b) exhibits three occurrences in the graph shown in (a), highlighted with dashed edges. Notice that two of those occurrences are on the same set of nodes, and the third one on a different set. (Note that occurrences can have different yet overlapping node sets, as in this example, as long as the edge sets are non-overlapping.) We refer to the number of occurrences of a motif g_i on the *same* set of nodes of a graph G_j, say $V \subseteq V_j$, as its *multiplicity on* V, denoted $m(g_i, G_j, V)$. Formally,

$$m(g_i, G_j, V) = \left|\left\{g_{ij}^{(k)} \mid V_{ij}^{(k)} = V\right\}\right| , \text{where } V \subseteq V_j \ . \tag{7}$$

Having provided all necessary definitions, Algo. 2 presents how a graph G_j is encoded. For an occurrence g_{kj} in its cover set, the corresponding motif g_k is first communicated by simply sending over its code-word (line 2). (Note that having encoded the motif table, the Receiver can translate the code-word to a specific motif structure.) Then, the one-to-one correspondence between the occurrence nodes V_{kj} and those of the motif V_k is encoded (line 3). Next, the multiplicity of the motif on the same set of nodes V_{kj} is encoded (line 4) so as to cover as many edges of G_j with few bits. Having been encoded, all those occurrences on V_{kj} are then removed from G_j's cover set (line 5).

Let us denote by $\overline{\mathcal{CS}_j} \subseteq \mathcal{CS}_j$ the *largest* subset of occurrences in $\mathcal{CS}_j$ with *unique* node-sets, such that $V_{kj} \neq V_{lj}, \forall \{g_{kj}, g_{lk}\} \in \overline{\mathcal{CS}_j}$. Then, the overall graph encoding length for G_j is given as

$$L(G_j|MT) = \sum_{g_{kj} \in \overline{\mathcal{CS}_j}} L(code_k) + L_{\text{perm}}(|V_j|, n_k) + L_{\mathbb{N}}(m(g_k, G_j, V_{kj})) \ . \tag{8}$$

The *usage* count of a motif $g_i \in \mathcal{M}$ that is used to compute its code-word *length* (Eq.s (2) and (3)) is defined as

$$usage_{\mathcal{G}}(g_i) = \sum_{G_j \in \mathcal{G}} c_{ij} = \sum_{G_j \in \mathcal{G}} \sum_{g_{kj} \in \mathcal{CS}_j} \mathbb{1}(g_{kj} \simeq g_i) \tag{9}$$

where $\mathbb{1}(g_{kj} \simeq g_i)$ returns 1 if g_{kj} is both structure- and label-isomorphic to g_i, that is if g_{kj} is an occurrence of g_i, and 0 otherwise. The basic idea is to encode each motif *only once* (as discussed in Sec. 4.1), and then to encode occurrences of a motif in the data $\mathcal{G}$ by simply providing a *reference* to the motif, i.e., its code-word. This way we can achieve compression by assigning high-occurrence motifs a *short* code-word, the principle behind Shannon's entropy.

5 SEARCH ALGORITHM

Our aim is to compress as a large portion of the input graphs as possible using motifs. This goal can be restated as finding a large set of non-overlapping motif occurrences that cover these graphs. We set up this problem as an instance of the Maximum Independent Set (*MIS*) problem on, what we call, the occurrence graph G_O. In G_O, the nodes represent motif occurrences and edges connect two occurrences that share a common edge. MIS ensures that occurrences in the solution are non-overlapping (thanks to independence, no two are incident to the same

Algorithm 2 Graph Encoding

Input: Graph G_j, Motif table MT, $\mathcal{CS}(G_j, MT)$
Output: Encoding of G_j such that it can be decoded losslessly

do
  Pick any occurrence $g_{kj} \in \mathcal{CS}(G_j, MT)$ and communicate $code_{MT}(g_k)$ ▷ $|code_{MT}(g_k)| = L(code_k)$ Eq. (2)
  Encode matching graph-node-IDs ▷ $L_{\text{perm}}(|V_j|, n_k)$ Eq. (6)
  Encode g_k's multiplicity on V_{kj} ▷ $L_{\mathbb{N}}(m(g_k, G_j, V_{kj}))$
  $\mathcal{CS}(G_j, MT) \leftarrow \mathcal{CS}(G_j, MT) \backslash \{g_{lj} \mid V_{lj} = V_{kj}\}$
while $\mathcal{CS}(G_j, MT) \neq \emptyset$

edge). Moreover, it helps us identify motifs that have large usages, i.e. number of non-overlapping occurrences (thanks to maximality), which is associated with shorter code length and hence better compression.

In this section, first we describe how we set up and solve the MIS problems, which provides us with a set of candidate motifs that can go into the MT as well as their (non-overlapping) occurrences in input graphs. We then present a search procedure for selecting a subset of motifs among those candidates to minimize the total encoding length in Eq. (1).

5.1 Step 1: Identifying Candidate Motifs & Their Occurrences

As a first attempt, we explicitly construct a G_O per input graph and solve *MIS* on it. Later, we present an efficient way for solving *MIS* without constructing any G_O's, which cuts down memory requirements drastically.

5.1.1 First Attempt: Constructing G_O's Explicitly.

Occurrence Graph Construction. For each $G_j \in \mathcal{G}$ we construct a $G_O = (V_O, E_O)$ as follows. For $k = 3, \ldots, 10$, we enlist all *connected* induced k-node subgraphs of G_j, each of which corresponds to an occurrence of some k-node motif in G_j. All those define the node set V_O of G_O. If any two enlisted occurrences share at least one common edge in G_j, we connect their corresponding nodes in G_O with an edge.

Notice that we do not explicitly enumerate all possible k-node labeled motifs and then identify their occurrences in G_j, if any, which would be expensive due to subgraph enumeration (esp. with many node labels) and numerous graph isomorphism tests. The above procedure yields occurrences of all *existing* motifs in G_j *implicitly*.

Greedy *MIS* Solution. For the *MIS* problem, we employ a greedy algorithm (Algo. 3) that sequentially selects the node with the minimum degree to include in the solution set O. It then removes it along with its neighbors from the graph until G_O is empty. Let $deg_{G_O}(v)$ denote the degree of node v in G_O (the initial one or the one after removing nodes from it along the course of the algorithm), and $\mathcal{N}(v) = \{u \in V_O | (u, v) \in E_O\}$ be the set of v's neighbors in G_O.

Approximation ratio: The greedy algorithm for *MIS* provides a Δ-approximation [18] where $\Delta = \max_{v \in V_O} deg_{G_O}(v)$. In our case Δ could be fairly large, e.g., in 1000s, as many occurrences overlap due to edge multiplicities. Here we strengthen this approximation ratio to $\min\{\Delta, \Gamma\}$ as follows, where Γ is around 10 in our data.

Theorem 5.1. *For MIS on occurrence graph $G_O = (V_O, E_O)$, the greedy algorithm in Algo. 3 achieves an approximation ratio of $\min\{\Delta, \Gamma\}$ where $\Delta = \max_{v \in V_O} deg_{G_O}(v)$ and Γ is the maximum number of edges in a motif that exists in the occurrence graph.*

Proof. Let $MIS(G_O)$ denote the number of nodes in the optimal solution set for *MIS* on G_O. We prove that

$$|O| \geq \frac{1}{\min\{\Delta, \Gamma\}} MIS(G_O). \tag{10}$$

Let $G_O(v)$ be the vertex-induced subgraph on G_O by the vertex set $\mathcal{N}_v = \{v\} \cup \mathcal{N}(v)$ when node v is selected by the greedy algorithm to include in O. $G_O(v)$ has $\{(u, w) \in E_O | u, w \in \mathcal{N}_v\}$ as the set of edges. In our greedy algorithm, when a node v is selected to the solution set O, v and all of its current neighbors are removed from G_O. Hence,

$$\forall u, v \in O, \mathcal{N}_u \cap \mathcal{N}_v = \emptyset.$$

In other words, all the subgraphs $G_O(v), v \in O$ are independent (they do not share any nodes or edges). Moreover, since the greedy algorithm runs until $V_O = \emptyset$ or all the nodes are removed from O, we also have

$$\cup_{v \in O} \mathcal{N}_v = V_O.$$

Then, we can derive the following upper-bound for the optimal solution based on the subgraphs $G_O(v), \forall v \in O$:

$$MIS(G_O) \leq \sum_{v \in O} MIS(G_O(v)). \tag{11}$$

This upper-bound of $MIS(G_O)$ can easily be seen by the fact that the optimal solution set in G_O can be decomposed into subsets where each subset is an independent set for a subgraph $G_O(v)$, hence, it is only a feasible solution of the *MIS* problem on that subgraph with a larger optimal solution.

Next, we will find an upper-bound for $MIS(G_O(v))$ and then plug it back in Eq. 11 to derive the overall bound. There are two different upper-bounds for $MIS(G_O(v))$ as follows:

Upper-bound 1: Assume that $E_O \neq \emptyset$, otherwise the greedy algorithm finds the optimal solution which is V_O and the theorem automatically holds, we establish the first bound:

$$\begin{aligned} MIS(G_O(v)) &\leq \max\{1, |\mathcal{N}(v)|\} = \max\{1, deg_{G_O}(v)\} \\ &\leq \max_{u \in V} deg_{G_O}(u) \leq \Delta \end{aligned} \tag{12}$$

Note that Δ is the maximum degree in the initial graph which is always larger or equal to the maximum degree at any point after removing nodes and edges.

Upper-bound 2: Let us consider the case that the optimal solution for *MIS* on $G_O(v)$ is the subset of $\mathcal{N}(v)$. Note that each node $v \in V_O$ is associated with an occurrence which contains a set of edges in G_j, denoted by $E_j(v) = \{e_{v1}, e_{v2}, \dots\}$, and, u and v are neighbors if $E_j(u) \cap E_j(v) \neq \emptyset$. Moreover, any pair of nodes u, w in the optimal independent set solution of $G_O(v)$ satisfies $E_j(u) \cap E_j(w) = \emptyset$. Thus, the largest possible independent set is $\{u_1, \dots, u_{|E_j(v)|}\}$ such that $E_j(u_l) \cap E_j(v) = \{e_{vl}\}$ (u_l must be a minimal neighbor - sharing a single edge between their corresponding occurrences) and $E_j(u_l) \cap E_j(u_k) = \emptyset, \forall l \neq k$ (every node is independent of the others). Therefore, we derive the second upper-bound:

$$MIS(G_O(v)) \leq |\{u_1, \dots, u_{|E_j(v)|}\}| = |E_j(v)| \leq \Gamma. \tag{13}$$

Combining Eq.s 12 and 13 gives us a stronger upper-bound:

$$MIS(G_O(v)) \leq \min\{\Delta, \Gamma\}. \tag{14}$$

Plugging Eq. 14 back to Eq. 11, we obtain

$$\begin{aligned} MIS(G_O) &\leq \sum_{v \in O} \min\{\Delta, \Gamma\} = |O| \times \min\{\Delta, \Gamma\}, \\ \Rightarrow\ |O| &\geq \frac{1}{\min\{\Delta, \Gamma\}} MIS(G_O). \end{aligned}$$

□

Algorithm 3 Greedy Algorithm for *MIS*

Input: Motif occurrence graph $G_O = (V_O, E_O)$ for graph G_j
Output: Set $O \subset V_O$ of independent nodes (non-overlapping occurrences)
1: Solution set $O = \emptyset$
2: **while** $V_O \neq \emptyset$ **do**
3: $v_{\min} = \arg\min_{v \in V_O} deg_{G_O}(v)$
4: $O := O \cup \{v_{\min}\}$
5: Remove $v_{\min}$ and $\mathcal{N}(v_{\min})$ along with incident edges from V_O and E_O accordingly

Complexity analysis: Algo. 3 requires finding the node with minimum degree (line 3) and removing nodes and incident edges (line 5), hence, a naïve implementation would have $O(|V_O|^2 + |E_O|)$ time complexity; $O(|V_O|^2)$ for searching nodes with minimum degree at most $|V_O|$ times (line 3) and $O(E_O)$ for updating node degrees (line 5). If we use a priority heap to maintain node degrees for a quick minimum search, time complexity becomes $O((|V_O| + |E_O|) \log |V_O|)$; $|V_O| \log |V_O|$ for constructing the heap initially and $|E_O| \log |V_O|$ for updating the heap every time a node and its neighbors are removed (includes deleting the degrees for the removed nodes and updating those of all $\mathcal{N}(v)$'s neighbors).

Algo. 3 takes $O(|V_O| + |E_O|)$ to store the occurrence graph.

5.1.2 **Memory-Efficient Solution: *MIS* w/out Explicit G_O.** Notice that the size of each G_O can be very large due to the combinatorial number of k-node subgraphs induced, which demands huge memory and time. Here we present an efficient version of the greedy algorithm that drastically cuts down the input size to MIS. Our new algorithm leverages a property stated as follows.

Property 1 (Occurrence Degree Equality). *The nodes in the occurrence graph G_O of G_j that correspond to occurrences that are enlisted based on subgraphs induced on the* same *node set in G_j have exactly the* same *degree.*

Let us introduce a new definition called *simple occurrence*. A simple occurrence $sg_{ij} = (V_{ij}, E_{ij})$ is a simple subgraph without the edge multiplicities of G_j that is isomorphic to a motif. Let $\{sg_{1j}, sg_{2j}, \ldots, sg_{tj}\}$ be the set of all simple occurrences in G_j. Note that two simple occurrences may correspond to the same motif.

Recall that the greedy algorithm only requires the *node degrees* in G_O. Since all the nodes corresponding to occurrences that "spring out" of a simple occurrence have the same degree (Property 1), we simply use the simple occurrence as a "compound node" in place of all those degree-equivalent occurrences.[3] As such, the nodes in G_O now correspond to simple occurrences *only*. The degree of each node (say, sg_{ij}) is calculated as follows.

$$deg_{G_O}(sg_{ij}) = \prod_{(u,v)\in E_{ij}} m(u,v) - \left[\prod_{(u,v)\in E_{ij}} \big(m(u,v) - 1\big) \right] - 1 + \sum_{\{sg_{lj} | E_{lj} \cap E_{ij} \neq \emptyset\}} \Big(\prod_{(u,v)\in E_{lj}\setminus E_{ij}} m(u,v) \Big) \times \Big(\prod_{(u,v)\in E_{lj}\cap E_{ij}} m(u,v) - \prod_{(u,v)\in E_{lj}\cap E_{ij}} \big(m(u,v) - 1\big) \Big), \tag{15}$$

where $m(u,v)$ is the multiplicity of edge (u,v) in G_j. The first line of Eq. (15) depicts the "internal degree" among the degree-equivalent occurrences that originate from sg_{ij}. The rest captures the "external degree" to other occurrences that have an overlapping edge.

Memory-Efficient Greedy *MIS* Solution. The detailed steps of our memory-efficient greedy *MIS* algorithm is given in Algo. 4. We first calculate degrees of all simple occurrences (line 2) and then sequentially select the

[3]Given sg_{ij}, the number of its degree-equivalent occurrences in G_j is the product of edge multiplicities $m(u,v)$ for $(u,v) \in E_{ij}$.

Algorithm 4 MEMORY-EFFICIENT GREEDY *MIS*

Input: Simple occurrences $sg_{1j}, \ldots, sg_{tj}$, edge multiplicities in G_j
Output: A list $\mathcal{S}$ of simple occurrences

1: $\mathcal{S} = \emptyset$
2: Calculate degrees of all simple occurrences $deg_{G_O}(sg_{ij}), \forall i = 1 \ldots t$ using Eq. (15)
3: $deg_{\max} = \max_{i=1\ldots t} deg_{G_O}(sg_{ij})$
4: **while** $\exists i \in \{1 \ldots t\}$ s.t. $deg_{G_O}(sg_{ij}) < deg_{\max} + 1$ **do**
5: $\quad i^* = \arg\min_{i \in \{1\ldots t\}} deg_{G_O}(sg_{ij})$
6: $\quad \mathcal{S} = \mathcal{S} \cup \{sg_{i^*j}\}$
7: $\quad m((u,v)) = m((u,v)) - 1\ ,\ \forall (u,v) \in E_{i^*j}$
8: $\quad$ **for** $l = 1 \ldots t, E_{lj} \cap E_{i^*j} \neq \emptyset$ **do**
9: $\quad\quad$ **if** $m((u,v)) > 0\ \forall (u,v) \in E_{lj}$ **then**
10: $\quad\quad\quad$ Recalculate $deg_{G_O}(sg_{lj})$ using Eq. (15)
11: $\quad\quad$ **else**
12: $\quad\quad\quad deg_{G_O}(sg_{lj}) = deg_{\max} + 1$

one with the minimum degree, denoted sg_{i^*j}, included in the solution list $\mathcal{S}$ (lines 5, 6). To account for this selection, we decrease multiplicities of all its edges $(u,v) \in E_{i^*j}$ by 1 (line 7) and recalculate the degrees of simple occurrences that overlap with sg_{i^*j} (lines 8-12). If one of those simple occurrences that has an intersecting edge set with that of sg_{i^*j} contains at least one edge with multiplicity equal to 0 (due to decreasing edge multiplicities in line 7), a special value of $deg_{\max} + 1$ (line 3) is assigned as its degree. This is to signify that this compound node contains no more occurrences, and it is not to be considered in subsequent iterations (line 4).

Notice that we need not even construct an occurrence graph in Algo. 4, which directly operates on the set of simple occurrences in G_j, computing and updating degrees based on Eq. (15). Note that the same simple occurrence could be picked more than once by the algorithm. The number of times a simple occurrence appears in $\mathcal{S}$ is exactly the number of non-overlapping occurrences that spring out of it and get selected by Algo. 3 on G_O. As such O and $\mathcal{S}$ have the same cardinality and each motif has the same number of occurrences and simple occurrences in O and $\mathcal{S}$, respectively. As we need the number of times each motif is used in the cover set of a graph (i.e., its usage), both solutions are equivalent.

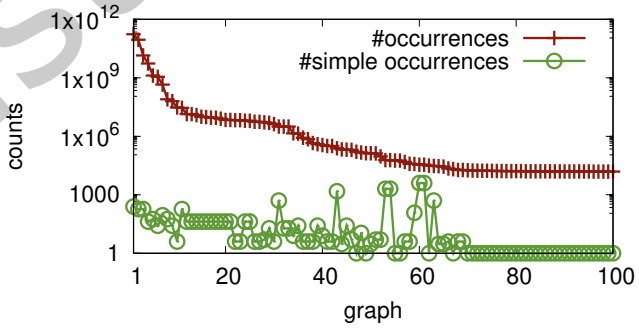


Fig. 2. Number of occurrences ($|V_O|$) and number of simple occurrences (t) for top 100 graphs in SH database with highest number of occurrences (in decreasing order). Note that the number of simple occurrences is up to 9 orders of magnitude smaller, thus, our memory-efficient algorithm drastically reduces the memory requirement by the same factor.

Complexity analysis: Calculating $deg_{G_O}(sg_{ij})$ of a simple occurrence takes $O(t \cdot \Gamma)$, as Eq. (15) requires all t intersecting simple occurrences $\{sg_{lj} \mid E_{lj} \cap E_{ij} \neq \emptyset\}$ where intersection can be done in $O(\Gamma)$, the maximum number of edges in a motif. Thus, Algo. 4 requires $O(t^2 \cdot \Gamma)$ for line 2 to calculate all degrees. Within the while loop (lines 4-12), the total number of degree recalculations (line 10) is bounded by $O(t \cdot \Gamma \cdot \max_{(u,v)\in E_j} m((u,v)))$ since each simple occurrence gets recalculated for at most $\Gamma \cdot \max_{(u,v)\in E_j} m((u,v))$ times. Finding the intersecting simple occurrences (line 8) is $(t \cdot \Gamma)$. Overall, Algo. 4 time complexity is $O(t^2 \cdot \Gamma^2 \cdot \max_{(u,v)\in E_j} m(u,v))$.

The space complexity of Algo. 4 is $O(t \cdot \Gamma + |E_j|)$, for t simple occurrences with at most Γ edges each, plus input edge multiplicities. Note that this is significantly smaller than the space complexity of Algo. 3, i.e. $O(|V_O| + |E_O|)$, since $t \ll |V_O|$ and $|E_j| \ll |E_O|$. Our empirical measurements on our SH dataset (see Table 2 in Sec. 6 for details) in Fig. 2 show that $|V_O|$ is up to 9 orders of magnitude larger than t.

5.1.3 **Weighted Maximum Independent Set (*WMIS*).** A motivation leading to our *MIS* formulation is to cover *as much of each input graph as possible* using motifs. The amount of coverage by an occurrence of a motif can be translated into the number of edges it contains. This suggests a *weighted* version of the maximum independent set problem, denoted *WMIS*, where we set the weight of an occurrence, i.e., node in G_O, to be the number of edges in the motif it corresponds to. Hence, the goal is to maximize the total weight of the non-overlapping occurrences in the solution set.

Greedy *WMIS* Solution. For the weighted version of *MIS*, we also have a greedy algorithm with the *same* approximation ratio as the unweighted one. The only difference from Algo. 3 is the selection of node $v_{\min}$ to remove (line 3). Let w_v denote the weight of node v in G_O. Then,

$$v_{\min} = \arg\max_{v \in V_O} \frac{w_v}{\min\{w_v, \deg_{G_O}(v)\} \cdot \max_{u \in \mathcal{N}(v)} w_u} \tag{16}$$

Intuitively, Eq. (16) prefers selecting large-weight nodes (to maximize total weight), but those that do not have too many large-weight neighbors in the G_O, which cannot be selected.

Theorem 5.2. *For WMIS on weighted occurrence graph $G_O = (V_O, E_O, W)$ where $w_v \in W$ is the weight of node $v \in V_O$, the greedy algorithm in Algo. 3 with node selection criterion in Eq.* (16) *achieves an approx. ratio of* $\min\{\Delta, \Gamma\}$ *where $\Gamma = \max_{v \in V_O} w_v$ is the maximum number of edges in a motif.*

Proof. The proof is similar to that of Theorem 5.1 with the key realization that the total weight of the optimal solution set of *WMIS* on subgraph $G_O(v)$ is upper-bounded by $\min\{w_v, \deg_{G_O}(v)\} \cdot \max_{u \in \mathcal{N}(v)} w_u$ and when a node v is selected from G_O by the greedy algorithm, due to the selection condition, we have,

$$\begin{aligned} \frac{w_v}{\min\{w_v, \deg_{G_O}(v)\} \cdot \max_{u \in \mathcal{N}(v)} w_u} &\geq \frac{w_{v^*}}{\min\{w_{v^*}, \deg_{G_O}(v^*)\} \cdot \max_{u \in \mathcal{N}(v^*)} w_u} \\ &\geq \frac{1}{\min\{w_{v^*}, \deg_{G_O}(v^*)\}} \geq \frac{1}{\min\{\Delta, \Gamma\}}, \end{aligned} \tag{17}$$

where $v^* = \max_{u \in V_O} w_u$, then $\forall u \in \mathcal{N}(v^*), w_{v^*} \geq w_u$. □ □

Memory-efficient Greedy *WMIS* Solution. Similar to the unweighted case, we can derive a memory-efficient greedy algorithm for *WMIS* since Property 1 holds for both degree $deg_{G_O}(v)$ and w_v—the two core components in selecting nodes (Eq. (16)) in greedy algorithm for *WMIS*—since all occurrences of the same simple occurrence have the same number of edges.

Algorithm 5 Motif Table Search

Input: Database $\mathcal{G} = \{G_1, \ldots, G_J\}$, candidate motifs C, node labels $\mathcal{T}$, $(W)MIS$ solution $\mathcal{S}_j$ $\forall G_j \in \mathcal{G}$
Output: MT containing set of motifs $\mathcal{M}$

1: $\text{Cnt}_j(s,d) = 0$, for $j = 1 \ldots J$, and $\forall s, d \in \mathcal{T}$
2: **for** $G_j = (V_j, E_j) \in \mathcal{G}$ **do**
3: **for each** $(u,v) \in E_j$ **do**
4: $\text{Cnt}_j(t(u), t(v)) := \text{Cnt}_j(t(u), t(v)) + m(u,v)$
5: $\mathcal{M} = \{(s,d) | s, d \in \mathcal{T}$ and $\exists j \in [1, J]$ where $\text{Cnt}_j(s,d) > 0\}$
6: **while** $C \neq \emptyset$ **do**
7: **for** $g = (V, E) \in C$ **do**
8: **for** $G_j \in \mathcal{G}$ **do**
9: $O(g, C\mathcal{S}_j) = \{sg \simeq g | sg \in \mathcal{S}_j\}$
10: $\text{Cnt}_{jg} := \text{Cnt}_j$
11: **for each** $(u,v) \in E$ **do**
12: $\text{Cnt}_{jg}(t(u), t(v)) := \text{Cnt}_{jg}(t(u), t(v)) - |O(g, C\mathcal{S}_j)|$
13: $\mathcal{M}_g = \mathcal{M} \cup \{g\} \backslash \{(s,d) | s, d \in \mathcal{T}, \text{Cnt}_{jg}(s,d) = 0 \; \forall j\}$
14: Get $L(\mathcal{M}_g, \mathcal{G})$ using $O(g_k, C\mathcal{S}_j)$ $\forall g_k \in \mathcal{M}_g$ s.t. $n_k \geq 3$ and $usage_{G_j}(s,d) := \text{Cnt}_{jg}(s,d)$ $\forall (s,d) \in \mathcal{M}_g; \forall j$
15: $g_{\min} = \arg\min_{g \in C} L(\mathcal{M}_g, \mathcal{G})$
16: **if** $L(\mathcal{M}_{g_{\min}}, \mathcal{G}) \leq L(\mathcal{M}, \mathcal{G})$ **then**
17: $\mathcal{M} := \mathcal{M}_{g_{\min}}$, $C := C \backslash \{g_{\min}\}$, $\text{Cnt}_j := \text{Cnt}_{jg_{\min}}$
18: **else**
19: **break**

5.2 Step 2: Building the Motif Table

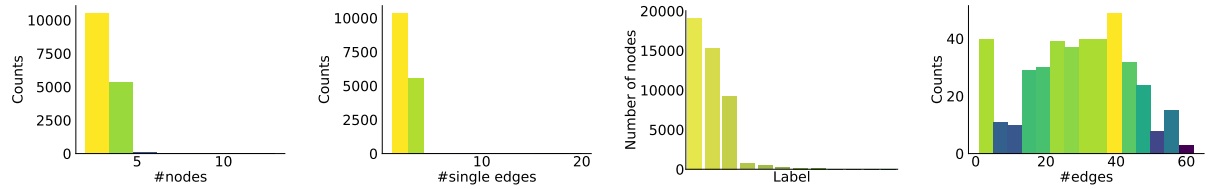
The $(W)MIS$ solutions, $\mathcal{S}_j$'s, provide us with non-overlapping occurrences of k-node motifs ($k \geq 3$) in each G_j. The next task is to identify the subset of those motifs to include in our motif table MT so as to minimize the total encoding length in Eq. (1). We first define the set C of *candidate motifs*:

$$C = \{g \simeq sg \mid sg \in \bigcup_j \mathcal{S}_j\}. \tag{18}$$

We start with encoding the graphs in $\mathcal{G}$ using the simplest code table that contains only the 2-node motifs. This code table, with optimal code lengths for database $\mathcal{G}$, is called the *Standard Code Table*, denoted by SMT. It provides the optimal encoding of $\mathcal{G}$ when nothing more is known than the frequencies of labeled edges (equal to *usage*s of the corresponding 2-node motifs), which are assumed to be fully independent. As such, SMT does not yield a good compression of the data but provides a practical bound.

To find a better code table, we use the best-first greedy strategy: Starting with $MT := SMT$, we try adding each of the candidate motifs in C one at a time. Then, we pick the 'best' one that leads to the largest reduction in the total encoding length. We repeat this process with the remaining candidates until no addition leads to a better compression or all candidates are included in the MT, in which case the algorithm terminates.

The details are given in Algo. 5. We first calculate the *usage* of 2-node motifs per G_j (lines 1-4), and set up the SMT accordingly (line 5). For each candidate motif $g \in C$ (line 7) and each G_j (line 8) we can identify the occurrences of g in G_j's cover set, $O(g, C\mathcal{S}_j)$, which is equivalent to the simple occurrences selected by Algo. 4 that are isomorphic to g (line 9). When we insert a g into MT, *usage* of some 2-node motifs, specifically those that correspond to the labeled edges of g, decreases by $|O(g, C\mathcal{S}_j)|$; the *usage* of g in G_j's encoding (lines 10-12). Note that the *usage* of $(k \geq 3)$-node motifs already in the MT do not get affected, since their occurrences in each $\mathcal{S}_j$, which we use to cover G_j, do *not* overlap; i.e., their uses in covering a graph are independent. As such,



(a) Distribution of the number of nodes.

(b) Distribution of the number of single edges.

(c) Distribution of node labels.

(d) Distribution of multiplicities for a pair of node labels.

Fig. 3. Statistical summary of graph characteristics in SH database

updating *usage*s when we insert a new motif to MT is quite efficient. Having inserted g and updated *usage*s, we remove 2-node motifs that reduce to zero *usage* from MT (line 13), and compute the total encoding length with the resulting MT (lines 14-15). The rest of the algorithm (lines 16-20) picks the 'best' g to insert that leads to the largest savings, if any, or otherwise quits.

6 EXPERIMENTS

Datasets. Our work is initiated by a collaboration with industry, and CODETECT is evaluated on large real-world datasets containing all transactions of 2016 (tens to hundreds of thousands transaction graphs) from 3 different companies, anonymously SH, HW, and KD (proprietary) summarized in Table 2. These do not come with ground truth anomalies. For quantitative evaluation, our expert collaborators inject two types of anomalies into each dataset based on domain knowledge (Sec. 6.1), and also qualitatively verify the detected anomalies from an accounting perspective (Sec. 6.2).

Since our transaction data are not shareable, to facilitate reproducibility of the results, we generate synthetic data that resemble the real-world datasets used. In particular, we add SH_SYNTHETIC that contains random graphs generated based on statistical characteristics of SH as follows: 1) A graph in SH_SYNTHETIC has the number of nodes randomly selected following its distribution in SH depicted in Figure 3a; 2) The number of single edges in a graph is drawn randomly following the distribution in SH illustrated in Figure 3b (If the graph has more edges than the maximum number - $n \times (n-1)/2$, we restart the process); 3) For each node, we randomly assign its label following the label distribution in SH as shown in Figure 3c; 4) For each single edge, we then randomly generate its multiplicity based on the distribution of the corresponding node label pair (an example is given in Figure 3d). At the end, we have the same number of graphs with similar characteristics as SH and share this data along with our code.

We also study the public Enron database [20], consisting of daily email graphs of its 151 employees over 3 years surrounding the financial scandal. Nodes depict employee email addresses and edges indicate email exchanges. Each node is labeled with the employee's department (Energy Operations, Regulatory and Government Affairs, etc.) and edge multiplicity denotes the number of emails exchanged.

6.1 Anomaly Detection Performance

We show that CODETECT is substantially better in detecting graph anomalies as compared to a list of baselines across various performance measures. The anomalies are injected by domain experts, which mimic schemes related to money laundering, entry error or malfeasance in accounting, specifically:

- *Path injection (money-laundering-like)*: (*i*) Delete a random edge $(u, v) \in E_j$, and (*ii*) Add a length- 2 or 3 path u–w(–z)–v where at least one edge of the path is rare (i.e., exists in only a few G_j's). The scheme mimics

Table 2. Summary statistics of graph datasets used in experiments.

Name	#Graphs	#Node labels	#Nodes	#Multi-edges
SH	38,122	11	[2, 25]	[1, 782]
SH_Synthetic	38,122	11	[2, 25]	[1, 782]
HW	90,274	11	[2, 25]	[1, 897]
KD	152,105	10	[2, 91]	[1, 1774]
Enron	1,139	16	[2, 87]	[1, 1356]

money-laundering, where money is transferred through multiple hops rather than directly from the source to the target.

- *Label injection (entry-error or malfeasance)*: (*i*) Pick a random node $u \in V_j$, and (*ii*) Replace its label $t(u)$ with a random label $t \neq t(u)$. This scheme mimics either simply an entry error (wrong account), or malfeasance that aims to reflect larger balance on certain types of account (e.g., revenue) in order to deceive stakeholders.

For path injection, we choose 3% of graphs and inject anomalous paths that replace 10% of edges (or 1 edge if 10% of edges is less than 1). For label injection, we also choose 3% of graphs and label-perturb 10% of the nodes (or 1 node if 10% of nodes is less than 1). We also tested with different severity levels of injection, i.e., 30% and 50% of edges or nodes, and observed similar results to those with 10%. The goal is to detect those graphs with injected paths or labels.

Baselines: We compare CODEtect with:

- SMT: A simplified version of CODEtect that uses the Standard Motif Table to encode the graphs.
- GLocalKD [26]: A recent graph neural network (GNN) based approach that leverages knowledge distillation to train and was shown to achieve good performance and robustness in semi-supervised and unsupervised settings. We used the default setting of 3 layers, 512 hidden dimensions, and 256 output dimensions as recommended in the original paper [26]. Independently, we also performed sensitivity analysis on varying the hyper-parameter settings and found that the default one returned top performance.
- Gbad [12]: The closest existing approach for anomaly detection in node-labeled graph databases (See Sec. 2). Since it cannot handle multi-edges, we input the G_j's as simple graphs setting all the edge multiplicities to 1.
- Graph Embedding + iForest: We pair different graph representation learning approaches with state-of-the-art outlier detector iForest [25], as they cannot directly detect anomalies. We consider the following combinations:
  - Graph2Vec [31] (G2V)+iForest: G2V cannot handle edge multiplicities, thus we set all to 1.
  - Deep Graph Kernel[54](DGK)+iForest
  - GF+iForest: Graph (numerical) features (GF) include number of edges of each label-pair and number of nodes of each label.
- Entropy quantifies skewness of the distribution on the non-zero number of edges over all possible label pairs as the anomaly score. A smaller entropy implies there exist rare label-pairs and hence higher anomalousness.
- Multiedges uses sum of edge multiplicities as anomaly score. We tried other simple statistics, e.g., #nodes, #edges, their sum/product, which do not perform well.

Performance measures: Based on the ranking of graphs by anomaly score, we measure Precision at top-k for $k = \{10, 100, 1000\}$, and also report Area Under ROC Curve (AUC) and Average Precision (AP) that is the area under the precision-recall curve. Since most of the methods, including CODEtect, are deterministic, we run most of the methods once and report all the measures. For Graph2Vec and Deep Graph Kernel with some

Table 3. Detection performance of *path anomalies* in various datasets. Numbers in **bold** highlight best results in the columns and underlined numbers refer to the runner-up. In all performance measures, CODETECT achieves significantly higher results than the others and maintains a large gap to the runner-up, which is not stable but varies depending on dataset and performance measure.

(a) SH dataset

Method	Prec@10	Prec@100	Prec@1000	AUC	AP
CODETECT	**1.000**	**0.920**	**0.386**	**0.958**	**0.548**
SMT	0.100	0.280	0.352	0.932	0.413
GLOCALKD	0.400	0.252	0.331	0.916	0.405
GBAD	0.200	0.495	0.356	0.906	0.373
GF+IFOREST	0.100	0.120	0.237	0.926	0.210
G2V+IFOREST	0.800	0.750	0.308	0.886	0.383
DGK+IFOREST	0.100	0.030	0.025	0.712	0.050
ENTROPY	0.100	0.800	0.219	0.821	0.347
MULTIEDGES	0.000	0.040	0.027	0.643	0.049

(b) HW dataset

Method	Prec@10	Prec@100	Prec@1000	AUC	AP
CODETECT	**0.900**	**0.990**	**0.999**	**0.995**	**0.772**
SMT	0.600	0.440	0.784	0.906	0.733
GLOCALKD	0.000	0.269	0.531	0.843	0.563
GBAD	0.800	0.710	0.685	0.930	0.555
GF+IFOREST	0.400	0.230	0.497	0.959	0.429
G2V+IFOREST	0.000	0.100	0.819	0.824	0.380
DGK+IFOREST	0.300	0.140	0.023	0.858	0.097
ENTROPY	0.300	0.820	0.896	0.981	0.571
MULTIEDGES	0.000	0.020	0.029	0.719	0.106

(c) SH_SYNTHETIC dataset

Method	Prec@10	Prec@100	Prec@1000	AUC	AP
CODETECT	**0.600**	**0.860**	**0.412**	**0.982**	**0.664**
SMT	0.300	0.440	0.212	0.911	0.398
GLOCALKD	0.300	0.495	0.258	0.932	0.473
GBAD	0.300	0.523	0.318	0.937	0.468
GF+IFOREST	0.200	0.420	0.252	0.894	0.340
G2V+IFOREST	0.500	0.640	0.310	0.943	0.450
DGK+IFOREST	0.100	0.030	0.025	0.712	0.050
ENTROPY	0.300	0.540	0.294	0.921	0.433
MULTIEDGES	0.000	0.050	0.036	0.662	0.102

Table 4. Detection performance of *label anomalies* in various datasets (**bold** and underlined numbers refer to the best and runner-up results). We observe notably higher performance of CODETECT compared to the competing baselines, where it ranks either at the top or is the runner-up across the board. In (b), – depicts cases for GBAD and G2V+IFOREST, which failed to complete within 5 days.

(a) HW dataset

Method	Prec@10	Prec@100	Prec@1000	AUC	AP
CODETECT	**0.800**	**0.720**	**0.709**	0.918	0.359
SMT	0.000	0.100	0.174	0.883	0.192
GLOCALKD	0.600	0.640	0.663	0.902	0.301
GBAD	**0.800**	0.710	0.685	**0.920**	**0.555**
GF+IFOREST	0.200	0.080	0.027	0.832	0.092
G2V+IFOREST	0.000	0.030	0.030	0.499	0.030
DGK+IFOREST	0.100	0.030	0.038	0.801	0.074
ENTROPY	0.100	0.030	0.117	0.623	0.062
MULTIEDGES	0.000	0.020	0.032	0.505	0.030

(b) KD dataset

Method	Prec@10	Prec@100	Prec@1000	AUC	AP
CODETECT	**0.800**	**0.940**	**0.863**	**0.716**	**0.403**
SMT	0.200	0.190	0.122	0.715	0.082
GLOCALKD	0.700	0.450	0.495	0.702	0.096
GBAD	–	–	–	–	–
GF+IFOREST	0.300	0.060	0.038	0.650	0.053
G2V+IFOREST	–	–	–	–	–
DGK+IFOREST	0.100	0.090	0.068	0.644	0.061
ENTROPY	0.400	0.240	0.130	0.541	0.040
MULTIEDGES	0.000	0.040	0.035	0.498	0.030

(c) SH_SYNTHETIC dataset

Method	Prec@10	Prec@100	Prec@1000	AUC	AP
CODETECT	**0.600**	**0.640**	**0.459**	**0.920**	**0.350**
SMT	0.400	0.450	0.162	0.720	0.163
GLOCALKD	0.400	0.440	0.321	0.874	0.289
GBAD	0.400	0.420	0.337	0.881	0.315
GF+IFOREST	0.200	0.250	0.140	0.831	0.127
G2V+IFOREST	0.000	0.020	0.021	0.465	0.020
DGK+IFOREST	0.200	0.110	0.063	0.739	0.125
ENTROPY	0.200	0.110	0.061	0.553	0.043
MULTIEDGES	0.000	0.014	0.024	0.475	0.020

randomization, we also run multiple times to check the consistency of performance. Additionally, for those with hyper-parameters, we use the default settings from the corresponding publicly available source codes.

6.1.1 **Detection of path anomalies.** We report detection results on SH and HW datasets in Tables 3a and 3b (performance on KD is similar and omitted for brevity) and on SH_Synthetic in Table 3c.

CODEtect consistently outperforms all baselines by a large margin across all performance measures in detecting path anomalies. More specifically, CODEtect provides 16.9% improvement over the runner-up (underlined) on average across all measures on SH, and 10.2% on HW. Note that the runner-up is not the same baseline consistently across different performance measures. Benefits of motif search is evident looking at the superior results over SMT. G2V+iForest produces decent performance w.r.t. most measures but is still much lower than those of CODEtect. Similar observations are present on SH_Synthetic.

6.1.2 **Detection of label anomalies.** Tables 4a and 4b report detection results on the two larger datasets, HW and KD (performance on SH is similar and omitted for brevity), and Table 4c provides results on SH_Synthetic. Note that Gbad and G2V+iForest failed to complete within 5 days on KD, thus their results are absent in Table 4b. Note that KD is a relatively large-scale dataset, having both larger and more graphs than the other datasets.

In general, we observe similar performance advantage of CODEtect over the baselines for label anomalies. The exceptions are Gbad and GLocalKD which perform comparably, and appear to suit better for label anomalies, potentially because changing node labels disrupts structure more than the addition of a few short isolate paths. Gbad however does not scale to KD, and the runner-up on this dataset performs significantly worse. Similar observations are also seen on SH_Synthetic dataset.

6.2 Case Studies

Case 1 - Anomalous transaction records: The original accounting databases we are provided with by our industry partner do not contain any ground truth labels. Nevertheless, they beg the question of whether CODEtect unearths any dubious journal entries that would raise an eyebrow from an economic bookkeeping perspective. In collaboration with their accounting experts, we analyze the top 20 cases as ranked by CODEtect. Due to space limit, we elaborate on one case study from each dataset/corporation as follows.

In SH, we detect a graph with a large encoding length yet relatively few (27) multi-edges, as shown in Fig. 5, consisting of several small disconnected components. In accounting terms the transaction is extremely complicated, likely the result of a (quite rare) "business restructuring" event. In this single journal entry there exist many independent simple entries, involving only one or two operating-expense (OE) accounts, while other edges arise from compound entries (involving more than three accounts). This event involves reversals (back to prepaid expenses) as well as re-classification of previously booked expenses. The fact that all these bookings are recorded within a single entry leaves room for manipulation of economic performance and mis-reporting via re-classification, which deserves an audit for careful re-examination.

In Fig. 4 (left) we show a motif with sole usage of 1 in the dataset, which is used to cover an anomalous graph (right) in HW. Edge NGL (non-operating gains&losses) to C (cash) depicts an unrealized foreign exchange gain and is quite unusual from an economic bookkeeping perspective. This is because, by definition, unrealized gains and losses do not involve cash. Therefore, proper booking of the creation or relinquishment of such gains or losses should not involve cash accounts. Another peculiarity is the three separate disconnected components, each of which represents very distinct economic transactions: one on a bank charge related to a security deposit, one on health-care and travel-related foreign-currency business expense (these two are short-term activities), and a third one on some on-going construction (long-term in nature). It is questionable why these diverse transactions are grouped into a single journal. Finally, the on-going construction portion involves reclassifying a long-term asset into a suspense account, which requires follow-up attention and final resolution.

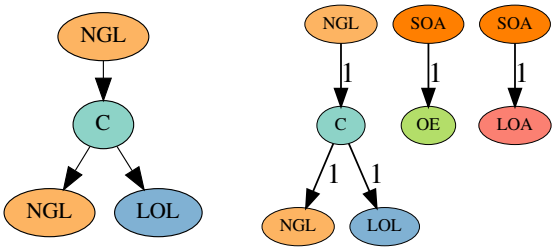


Fig. 4. (left) A rare motif, (right) Anomalous graph in HW.

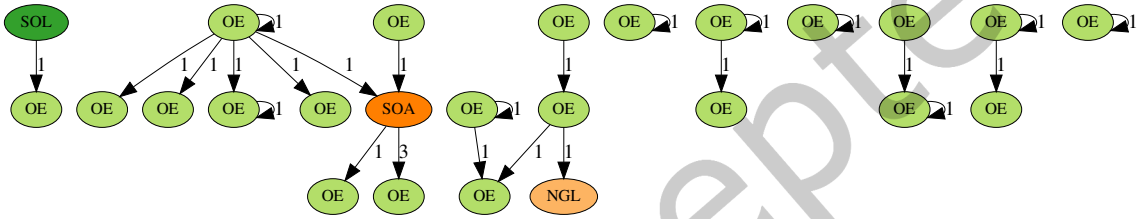


Fig. 5. Anomalous graph in SH.

Finally, the anomalous journal entry from KD involves the motif shown in Fig. 6 (left) where the corresponding graph is the exact motif with multiplicity 1 shown on the (right). This motif has sole usage of 1 in the dataset and is odd from an accounting perspective. Economically, it represents giving up an existing machine, which is a long-term operating asset (LOA), in order to reduce a payable or an outstanding short-term operating liability (SOL) owed to a vendor. Typically one would sell the machine and get cash to payoff the vendor with some gains or losses. We also note that the *MT* does not contain the 2-node motif LOA→SOL. The fact that it only shows up once, within single-usage motif, makes it suspicious.

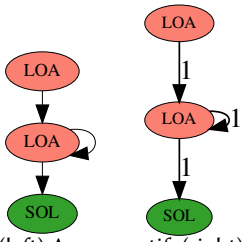
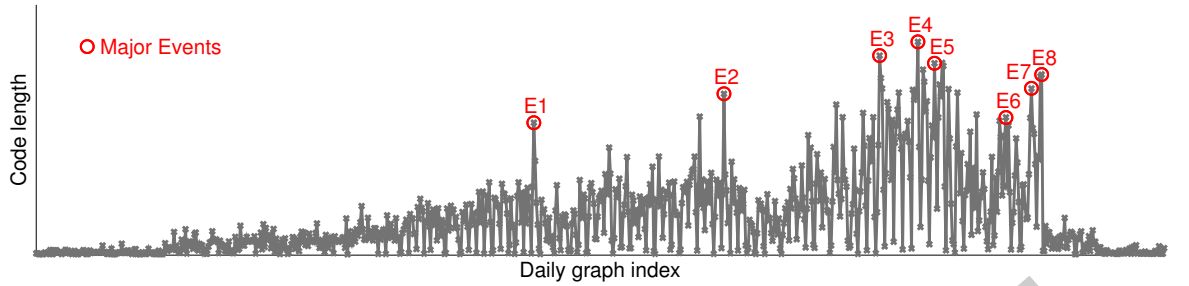


Fig. 6. (left) A rare motif, (right) Anomalous graph in KD.

Besides the quantitative evidence on detection performance, these number of case studies provide qualitative support to the effectiveness of CODETECT in identifying anomalies of interest in accounting domain terms, worthy of auditing and re-examination.

Case 2 - Enron scandal: We study the correlation between CODETECT's anomaly scores of the daily email exchange graphs and the major events in Enron's timeline. Fig. 7 shows that days with large anomaly scores mark drastic discontinuities in time, which coincide with important events related to the financial scandal[4]. It is

[4]http://www.agsm.edu.au/bobm/teaching/BE/Enron/timeline.html, https://www.theguardian.com/business/2006/jan/30/corporatefraud.enron



E1	Dec 13, 2000	Enron announces that president and COO Jeffrey Skilling will take over as chief executive in February.
E2	May 23, 2001	Enron completes its 1,000,000-th transaction via Enron Online.
E3	Sep 26, 2001	Employee Meeting. Kenneth Lay tells employees: Enron stock is an "incredible bargain" and "Third quarter is looking great."
E4	Oct 24-29, 2001	Enron sacks Andrew Fastow. In vain Lay calls chairman of the Fed and the Treasury and Commerce secretaries to solicit help.
E5	Nov 9, 2001	Dynegy agrees to buy Enron for about $9 billion in stock and cash.
E6	Jan 10, 2002	DOJ confirms criminal investigation begun. Arthur Andersen announces employees in Houston Division had destroyed documents.
E7	Jan 30, 2002	Enron names Stephen F. Cooper new CEO.
E8	Feb 07, 2002	A. Fastow invokes the Fifth Amendment before Congress; J. Skilling testifies he knew of no wrongdoing at Enron when he resigned.

Fig. 7. Code lengths of Enron's daily email exchange graphs. Large values coincide with key events of the financial scandal.

also noteworthy that the anomaly scores follow an increasing trend over days, capturing the escalation of events up to key personnel testifying in front of Congressional committees.

6.3 Scalability

To showcase the scalability of CODEtect, in regard to running time and memory consumption, we randomly selected subsets of graphs in KD database with different sizes, i.e., $\{40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100\} \times 10^3$. We re-sample each subset of graphs three times and report the averaged result for each setting. A summary of the results is presented in Figure 8. We observe a linear scaling of CODEtect with increasing size of input graphs as measured in number of multi-edges with respect to both time and memory usage.

7 CONCLUSION

We introduced CODEtect, (to our knowledge) the first graph-level anomaly detection method for *node-labeled multi-graph databases*; which appear in numerous real-world settings such as social networks and financial transactions, to name a few. The main idea is to identify key network motifs that encode the database concisely and employ compression length as the anomaly score. To this end, we presented (1) novel lossless encoding schemes and (2) efficient search algorithms. Experiments on transaction databases from three different corporations quantitatively showed that CODEtect significantly outperforms the prior and more recent GNN based baselines across datasets and performance metrics. Case studies, including the Enron database, presented qualitative evidence to CODEtect's effectiveness in spotting instances that are noteworthy of auditing and re-examination.

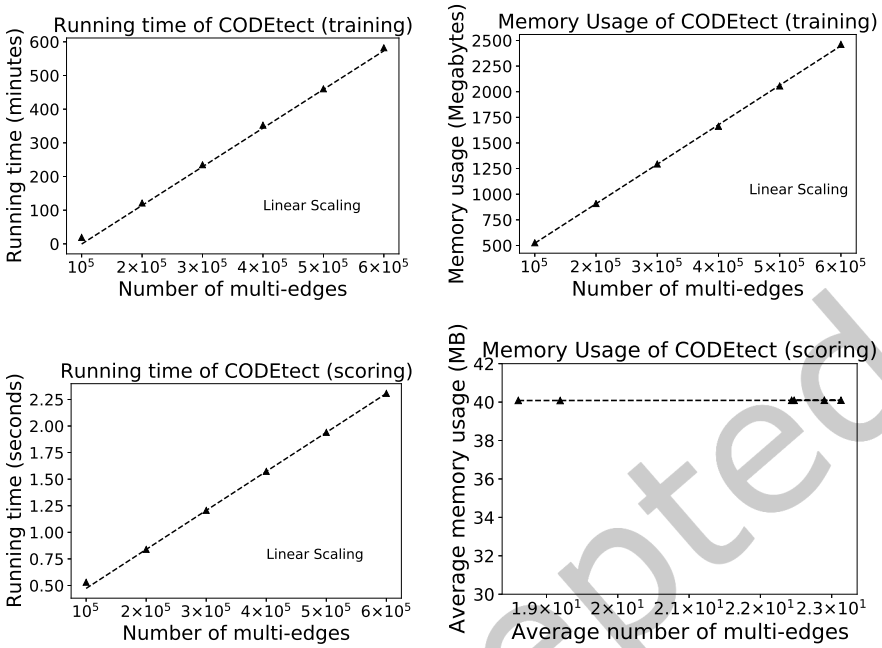


Fig. 8. (left) Time, (right) Memory consumption (training refers to running the full motif table search, while scoring assumes the motif table has been learned and only executes graph encoding for anomaly scoring.) Note that CODETECT training and scoring both scale linearly in the number of multi-edges w.r.t. both time and memory usage.

ACKNOWLEDGMENTS

This work is sponsored by NSF CAREER 1452425 and the PwC Risk and Regulatory Services Innovation Center at Carnegie Mellon University. Any conclusions expressed in this material are those of the author and do not necessarily reflect the views, expressed or implied, of the funding parties.

REFERENCES

[1] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. 2010. oddball: Spotting Anomalies in Weighted Graphs.. In *PAKDD*, Vol. 6119. Springer, 410–421.

[2] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data Min. Knowl. Discov.* 29, 3 (2015), 626–688.

[3] Leman Akoglu, Hanghang Tong, Jilles Vreeken, and Christos Faloutsos. 2012. Fast and reliable anomaly detection in categorical data.. In *CIKM*. 415–424.

[4] Mohammad Al Hasan and Vachik S. Dave. 2018. Triangle counting in large networks: a review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8, 2 (2018), e1226.

[5] A. Arenas, A. Fernández, S. Fortunato, and S. Gómez. 2008. Motif-based communities in complex networks. *Journal of Physics A* 41, 22 (2008), 224001.

[6] Sambaran Bandyopadhyay, Saley Vishal Vivek, and MN Murty. 2020. Outlier resistant unsupervised deep architectures for attributed network embedding. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 25–33.

[7] Austin R. Benson, David F. Gleich, and Jure Leskovec. 2016. Higher-order organization of complex networks. *Science* 353, 6295 (2016), 163–166.

[8] Peter Bloem and Steven de Rooij. 2017. Large-Scale Network Motif Learning with Compression. *CoRR* abs/1701.02026 (2017). arXiv:1701.02026

[9] Peter Bloem and Steven de Rooij. 2020. Large-scale network motif analysis using compression. *Data Mining and Knowledge Discovery* 34, 5 (2020), 1421–1453.

[10] Diane J. Cook and Lawrence B. Holder. 1994. Substructure Discovery Using Minimum Description Length and Background Knowledge. *JAIR* 1 (1994), 231–255.
[11] Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. 2019. Deep anomaly detection on attributed networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 594–602.
[12] William Eberle and Lawrence B. Holder. 2007. Anomaly detection in data represented as graphs. *Intell. Data Anal.* 11, 6 (2007), 663–689.
[13] Ethan R. Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G. Dimakis. 2015. Beyond Triangles: A Distributed Framework for Estimating 3-profiles of Large Graphs.. In *KDD*, Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams (Eds.). ACM, 229–238.
[14] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. 2018. SpotLight: Detecting Anomalies in Streaming Graphs.. In *KDD*. ACM, 1378–1386.
[15] Yujie Fan, Shifu Hou, Yiming Zhang, Yanfang Ye, and Melih Abdulhayoglu. 2018. Gotcha-Sly Malware! Scorpion A Metagraph2vec Based Malware Detection System. In *KDD*. 253–262.
[16] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. 855–864.
[17] Peter D Grünwald. 2007. *The minimum description length principle*. MIT press.
[18] Magnús M Halldórsson and Jaikumar Radhakrishnan. 1997. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica* 18, 1 (1997), 145–163.
[19] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1024–1034.
[20] Bryan Hooi, Kijung Shin, Hyun Ah Song, Alex Beutel, Neil Shah, and Christos Faloutsos. 2017. Graph-based fraud detection in the face of camouflage. *TKDD* 11, 4 (2017), 1–26.
[21] Zahra Kashani, Hayedeh Ahrabian, Elahe Elahi, Abbas Nowzari-Dalini, Elnaz Ansari, Sahar Asadi, Shahin Mohammadi, Falk Schreiber, and Ali Masoudi-Nejad. 2009. Kavosh: a new algorithm for finding network motifs. *BMC Bioinformatics* 10, 1 (2009), 318.
[22] Nadav Kashtan, Shalev Itzkovitz, Ron Milo, and Uri Alon. 2004. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics* 20, 11 (2004), 1746–1758.
[23] Danai Koutra, U. Kang, Jilles Vreeken, and Christos Faloutsos. 2014. VOG: Summarizing and Understanding Large Graphs.. In *SDM*. SIAM, 91–99.
[24] Lauri Kovanen, Márton Karsai, Kimmo Kaski, János Kertész, and Jari Saramäki. 2011. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics: Theory and Experiment* 2011, 11 (2011), P11005.
[25] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-based anomaly detection. *TKDD* 6, 1 (2012), 3.
[26] Rongrong Ma, Guansong Pang, Ling Chen, and Anton van den Hengel. 2022. Deep Graph-level Anomaly Detection by Glocal Knowledge Distillation. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 704–714.
[27] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z Sheng, Hui Xiong, and Leman Akoglu. 2021. A comprehensive survey on graph anomaly detection with deep learning. *IEEE Transactions on Knowledge and Data Engineering* (2021).
[28] Emaad A. Manzoor, Sadegh M. Milajerdi, and Leman Akoglu. 2016. Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs.. In *KDD*. ACM, 1035–1044.
[29] Ron Milo, Shalev Itzkovitz, Nadav Kashtan, Reuven Levitt, Shai Shen-Orr, Inbal Ayzenshtat, Michal Sheffer, and Uri Alon. 2004. Superfamilies of evolved and designed networks. *Science* 303, 5663 (2004), 1538–1542.
[30] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network Motifs: Simple Building Blocks of Complex Networks. *Science* 298, 5594 (2002), 824–827.
[31] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005* (2017).
[32] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. 2008. Graph summarization with bounded error.. In *SIGMOD*. ACM, 419–432.
[33] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *ICML*. 2014–2023.
[34] Caleb C. Noble and Diane J. Cook. 2003. Graph-based anomaly detection.. In *KDD*. ACM, 631–636.
[35] Ashwin Paranjape, Austin R. Benson, and Jure Leskovec. 2017. Motifs in Temporal Networks. In *WSDM*. ACM, 601–610.
[36] Ramesh Paudel and William Eberle. 2020. SNAPSKETCH: Graph Representation Approach for Intrusion Detection in a Streaming Graph. In *MLG 2020: 16th International Workshop on Mining and Learning with Graphs*. ACM.
[37] Hao Peng, Jianxin Li, Qiran Gong, Senzhang Wang, Yuanxing Ning, and Philip S Yu. 2018. Graph convolutional neural networks via motif-based attention. *arXiv preprint arXiv:1811.08270* (2018).
[38] Bryan Perozzi and Leman Akoglu. 2016. Scalable anomaly ranking of attributed neighborhoods. In *SDM*. SIAM, 207–215.
[39] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. 2014. Focused clustering and outlier detection in large attributed graphs. In *KDD*. 1346–1355.
[40] Jorma Rissanen. 1978. Modeling by shortest data description. *Automatica* 14, 5 (1978), 465–471.
[41] Ryan A. Rossi, Nesreen K. Ahmed, Aldo G. Carranza, David Arbour, Anup Rao, Sungchul Kim, and Eunyee Koh. 2019. Heterogeneous Network Motifs. *CoRR* abs/1901.10026 (2019). arXiv:1901.10026

[42] Seyed-Vahid Sanei-Mehri, Ahmet Erdem Sariyüce, and Srikanta Tirthapura. 2018. Butterfly Counting in Bipartite Networks.. In *KDD*, Yike Guo and Faisal Farooq (Eds.). ACM, 2150–2159.
[43]]SeTi19 C. Seshadhri and Srikanta Tirthapura. [n. d.]. Scalable Subgraph Counting: The Methods Behind The Madness: WWW 2019 Tutorial. In *WWW*.
[44] Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. 2015. TimeCrunch: Interpretable Dynamic Graph Summarization.. In *KDD*. ACM, 1055–1064.
[45] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *AISTATS*. 488–495.
[46] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. 2016. Corescope: Graph mining using k-core analysis—patterns, anomalies and algorithms. In *ICDM*. IEEE, 469–478.
[47] Koen Smets and Jilles Vreeken. 2011. The Odd One Out: Identifying and Characterising Anomalies.. In *SDM*. SIAM / Omnipress, 804–815.
[48] Nikolaj Tatti and Jilles Vreeken. 2008. Finding Good Itemsets by Packing Data.. In *ICDM*. IEEE Computer Society, 588–597.
[49] Yuanyuan Tian, Richard A. Hankins, and Jignesh M. Patel. 2008. Efficient aggregation for graph summarization.. In *SIGMOD*. ACM, 567–580.
[50] Johan Ugander, Lars Backstrom, and Jon M. Kleinberg. 2013. Subgraph frequencies: mapping the empirical and extremal geography of large graph collections.. In *WWW*, Daniel Schwabe, Virgílio A. F. Almeida, Hartmut Glaser, Ricardo A. Baeza-Yates, and Sue B. Moon (Eds.). ACM, 1307–1318.
[51] A. Vázquez, R. Dobrin, D. Sergi, J.-P. Eckmann, Z. N. Oltvai, and A.-L. Barabási. 2004. The topological relationship between the large-scale attributes and local interaction patterns of complex networks. *PNAS* 101, 52 (2004), 17940–17945.
[52] Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. 2011. Krimp: mining itemsets that compress. *Data Min. Knowl. Discov.* 23, 1 (2011), 169–214.
[53] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.
[54] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *KDD*. 1365–1374.
[55] Hao Yin, Austin R. Benson, and Jure Leskovec. 2018. Higher-order clustering in networks. *Phys. Rev. E* 97 (2018), 052306.
[56] Lingxiao Zhao and Leman Akoglu. 2021. On using classification datasets to evaluate graph outlier detection: Peculiar observations and new insights. *Big Data* (2021).
[57] Tong Zhao, Chuchen Deng, Kaifeng Yu, Tianwen Jiang, Daheng Wang, and Meng Jiang. 2020. Error-Bounded Graph Anomaly Loss for GNNs. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1873–1882.