Provably Powerful Graph Neural Networks for Directed Multigraphs

Béni Egressy[1]*, Luc von Niederhäusern[1], Jovan Blanuša[2], Erik Altman[3], Roger Wattenhofer[1], Kubilay Atasu[2]

[1]ETH Zurich, Zurich, Switzerland
[2]IBM Research Europe, Zurich, Switzerland
[3]IBM Watson Research, Yorktown Heights, NY, USA
begressy@ethz.ch, lucv@ethz.ch, jov@zurich.ibm.com, ealtman@us.ibm.com, wattenhofer@ethz.ch, kat@zurich.ibm.com

Abstract

This paper analyses a set of simple adaptations that transform standard message-passing Graph Neural Networks (GNN) into provably powerful directed multigraph neural networks. The adaptations include multigraph port numbering, ego IDs, and reverse message passing. We prove that the combination of these theoretically enables the detection of any directed subgraph pattern. To validate the effectiveness of our proposed adaptations in practice, we conduct experiments on synthetic subgraph detection tasks, which demonstrate outstanding performance with almost perfect results.

Moreover, we apply our proposed adaptations to two financial crime analysis tasks. We observe dramatic improvements in detecting money laundering transactions, improving the minority-class F1 score of a standard message-passing GNN by up to 30%, and closely matching or outperforming tree-based and GNN baselines. Similarly impressive results are observed on a real-world phishing detection dataset, boosting three standard GNNs' F1 scores by around 15% and outperforming all baselines. To appear as a conference paper at AAAI 2024.

1 Introduction

Graph neural networks (GNNs) have become the go-to machine learning models for learning from relational data. GNNs are used in various fields, ranging from biology, physics, and chemistry to social networks, traffic, and weather forecasting (Bongini, Bianchini, and Scarselli 2021; Zhou et al. 2020; Derrow-Pinion et al. 2021; Shu, Wang, and Liu 2019; Wu et al. 2020; Keisler 2022; Zhang et al. 2019; Battaglia et al. 2016). More recently, there has been growing interest in using GNNs to identify financial crime (Cardoso, Saleiro, and Bizarro 2022; Kanezashi et al. 2022; Weber et al. 2019, 2018; Nicholls, Kuppa, and Le-Khac 2021).

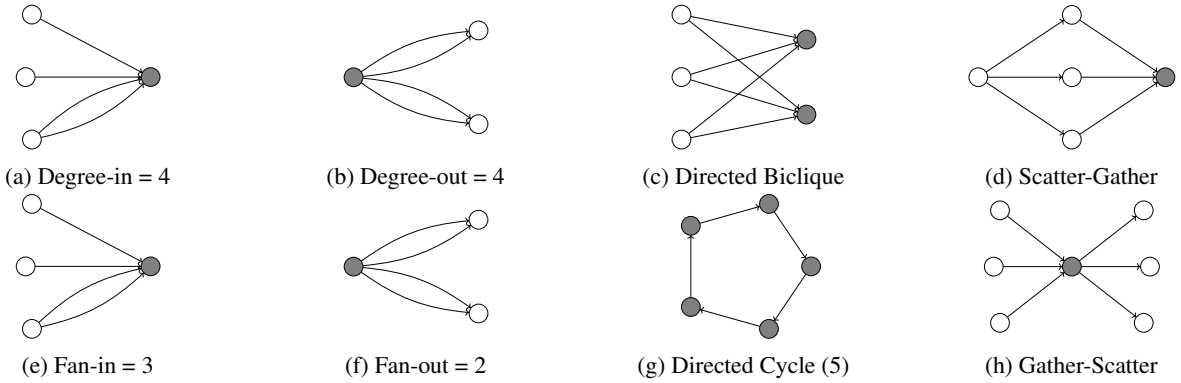
Our motivating task is to detect financial crimes manifesting as subgraph patterns in transaction networks. For example, see Fig. 1, which depicts established money laundering patterns, or see Fig. 4 in the appendix for an illustrative scenario. But note that similar patterns are relevant for graph tasks in many areas, ranging from chemistry to traffic forecasting. The task seems to lend itself nicely to the use of GNNs. Unfortunately, current GNNs are ill-equipped to deal with financial transaction networks effectively.

Firstly, financial transaction networks are, in fact, directed multigraphs, i.e., edges (or transactions) have a direction, and there can be multiple edges between two nodes (or accounts). Secondly, most GNNs cannot detect some subgraph patterns, such as cycles (Chen et al. 2020, 2019b). There have been many efforts to overcome this limitation (You et al. 2021; Huang et al. 2022; Papp and Wattenhofer 2022; Zhang and Li 2021; Loukas 2019; Sato, Yamada, and Kashima 2019), all focusing on simple (undirected) graphs. But even on simple graphs, the problem is far from solved. Until very recently, for example, there was no linear-time permutation-equivariant GNN that could count 6-cycles with theoretical guarantees (Huang et al. 2022).

This paper addresses both of these issues. To our knowledge, this is the first GNN architecture designed specifically for directed multigraphs. Secondly, we first prove that the proposed architecture can theoretically detect any subgraph pattern in directed multigraphs and then empirically confirm that our proposed architecture can detect the patterns illustrated in Fig. 1. Our proposed architecture is based on a set of simple adaptations that can transform any standard GNN architecture into a directed multigraph GNN. The adaptations are reverse message passing (Jaume et al. 2019), port numbering (Sato, Yamada, and Kashima 2019), and ego IDs (You et al. 2021). Although these individual building blocks are present in existing literature, the theoretical and empirical power of combining them has not been explored. In this work, we fill this gap: We combine them, adapt them to directed multigraphs, and showcase the theoretical and empirical advantages of using them in unison.

Our contributions. (1) We propose a set of simple and intuitive adaptations that can transform message-passing GNNs into provably powerful directed multigraph neural networks. (2) We prove that suitably powerful GNNs equipped with ego IDs, port numbering, and reverse message passing can identify any directed subgraph pattern. (3) The theory is tested on synthetic graphs, confirming that GNNs using these adaptations can detect a variety of subgraph patterns, including directed cycles up to length six, scatter-gather patterns, and directed bicliques, setting them apart from previous GNN architectures. (4) The improvements translate to significant gains on two financial datasets. The adaptations boost GNN

*This work was performed while Béni Egressy and Luc von Niederhäusern were at IBM Research Europe, Zurich, Switzerland.



(a) Degree-in = 4 (b) Degree-out = 4 (c) Directed Biclique (d) Scatter-Gather

(e) Fan-in = 3 (f) Fan-out = 2 (g) Directed Cycle (5) (h) Gather-Scatter

Figure 1: Money Laundering Patterns. The gray fill indicates the nodes to be detected by the synthetic pattern detection tasks. The exact degree/fan pattern sizes here are for illustrative purposes only.

performance dramatically on money laundering and phishing datasets, matching or surpassing state-of-the-art financial crime detection models on both simulated and real data.

2 Related Work

Xu et al. (2018) showed that standard MPNNs are at most as powerful as the Weisfeiler-Lehman (WL) isomorphism test, and provided a GNN architecture, GIN, that theoretically matches the power of the WL test. Although the WL test can asymptotically almost surely differentiate any two non-isomorphic graphs (Babai, Erdos, and Selkow 1980), standard MPNNs cannot — in certain graphs — detect simple substructures like cycles (Chen et al. 2020, 2019b). This motivated researchers to go beyond standard MPNNs.

One direction considers emulating the more powerful k-WL isomorphism test, by conducting message passing between k-tuples or using a tensor-based model (Maron et al. 2019; Morris et al. 2019). Unfortunately, these models have high complexity and are impractical for most applications. Another line of work uses pre-calculated features to augment the GNN. These works explore adding subgraph counts (Bouritsas et al. 2022; Barceló et al. 2021), positional node embeddings (Egressy and Wattenhofer 2022; Dwivedi et al. 2021), random IDs (Abboud et al. 2020; Sato, Yamada, and Kashima 2021), and node IDs (Loukas 2019).

A recent class of expressive GNNs called Subgraph GNNs, model graphs as collections of subgraphs (Frasca et al. 2022; Zhao et al. 2021). Papp et al. (2021) drop random nodes from the input and run the GNN multiple times, gathering more information with each run. Zhang and Li (2021) instead extract subgraphs around each node and run the GNN on these. Also falling into this category is ID-GNN, which uses ego IDs (You et al. 2021), whereby each node is sampled with its neighborhood and given an identifier to differentiate it from the neighbors. Although the authors claim that ID-GNNs can count cycles, the proof turns out to be incorrect. In fact, Huang et al. (2022) show that the whole family of Subgraph GNNs cannot count cycles of length greater than 4, and propose I^2-GNNs that can count cycles up to length 6.

There has been much less work on GNNs for directed graphs. Zhang et al. (2021) propose a spectral network for directed graphs, but it is difficult to analyze the power of this network or apply it to larger datasets. Similar approaches can be found in (Tong et al. 2020) and (Ma et al. 2019). Jaume et al. (2019) extend message passing to aggregate incoming and outgoing neighbors separately, rather than naively treating the graph as undirected. Directed multigraphs have not specifically been considered.

GNNs have been used for various financial applications (Li et al. 2021; Feng et al. 2019; Chen, Wei, and Huang 2018; Zhang et al. 2019; Li et al. 2019; Xu et al. 2021; Yang et al. 2021). Closest to our work, GNNs have been used for fraud detection. Liang et al. (2019) and Rao et al. (2021) work on bipartite customer-product graphs to uncover insurance and credit card fraud, respectively. Liu et al. (2018) use heterogeneous GNNs to detect malicious accounts in the device-activity bipartite graph of an online payment platform. Weber et al. (2019) were the first to apply standard GNNs for anti-money laundering (AML), and more recently Cardoso, Saleiro, and Bizarro (2022) proposed representing the transaction network as a bipartite account-transaction graph and showed promising results in the semi-supervised AML setting. However, it is not clear how these approaches help with detecting typical fraud patterns.

3 Background

3.1 Graphs and Financial Transaction Graphs

We consider directed multigraphs, G, where the nodes $v \in V(G)$ represent accounts, and the directed edges $e = (u, v) \in E(G)$ represent transactions from u to v. Each node u (optionally) has a set of account features $h^{(0)}(u)$; this could include the account number, bank ID, and account balance. Each transaction $e = (u, v)$ has a set of associated transaction features $h^{(0)}_{(u,v)}$; this includes the amount, currency, and timestamp of the transaction. The incoming and outgoing neighbors of u are denoted by $N_{in}(u)$ and $N_{out}(u)$ respectively. Multiple transactions between the same two accounts are possible, making G a multigraph. In node (or edge) prediction tasks, each node (or edge) will have a binary label indicating whether the account (or transaction) is illicit.

Financial Crime Patterns. Fig. 1 shows a selection of

subgraph patterns indicative of money laundering (Granados and Vargas 2022; He et al. 2021; Suzumura 2022; Weber et al. 2018; Starnini et al. 2021). Unfortunately, these are rather generic patterns, which also appear extensively amongst perfectly innocent transactions. As a result, detecting financial crime relies not just on detecting individual patterns, but also on learning relevant combinations. This makes neural networks promising candidates for the task. However, standard message-passing GNNs typically fail to detect the depicted patterns, except for degree-in. In the next section, we describe architectural adaptations, which enable GNNs to detect each one of these patterns.

Subgraph Detection. Given a subgraph pattern H, we define subgraph detection for nodes as deciding for each node in a graph whether it is part of a subgraph that is isomorphic to H; i.e., given a node $v \in V(G)$, deciding whether there exists a graph G', with $E(G') \subseteq E(G)$ and $V(G') \subseteq V(G)$, such that $v \in V(G')$ and $G' \cong H$.

3.2 Message Passing Neural Networks

Message-passing GNNs, commonly referred to as Message Passing Neural Networks (MPNNs), form the most prominent family of GNNs. They include GCN (Kipf and Welling 2016), GIN (Xu et al. 2018), GAT (Veličković et al. 2017), GraphSAGE (Hamilton, Ying, and Leskovec 2017), and many more architectures. They work in three steps: (1) Each node sends a message with its current state $h(v)$ to its neighbors, (2) Each node aggregates all the messages it receives from its neighbors in the embedding $a(v)$, and (3) Each node updates its state based on $h(v)$ and $a(v)$ to produce a new state. These 3 steps constitute a layer of the GNN, and they can be repeated to gather information from further and further reaches of the graph. More formally:

$$a^{(t)}(v) = \textsc{Aggregate}\left(\{h^{(t-1)}(u) \mid u \in N(v)\}\right),$$

$$h^{(t)}(v) = \textsc{Update}\left(h^{(t-1)}(v), a^{(t)}(v)\right),$$

where $\{\{.\}\}$ denotes a multiset, and AGGREGATE is a permutation-invariant function. We will shorten AGGREGATE to AGG, and for readability, we will use $\{.\}$ rather than $\{\{.\}\}$ to indicate multisets.

In the case of directed graphs, we need to distinguish between the incoming and outgoing neighbors of node u. In a standard MPNN, the messages are passed along the directed edges in the direction indicated. As such, the aggregation step only considers messages from incoming neighbors:

$$a^{(t)}(v) = \textsc{Agg}\left(\{h^{(t-1)}(u) \mid u \in N_{in}(v)\}\right),$$

where we aggregate over the incoming neighbors, $N_{in}(v)$.

The edges of an input graph may also have input features. We denote the input features of directed edge $e = (u, v)$ by $h^{(0)}((u, v))$. When using edge features during the message passing, the aggregation step becomes:

$$a^{(t)}(v) = \textsc{Agg}\left(\{(h^{(t-1)}(u), h^{(0)}((u, v))) \mid u \in N_{in}(v)\}\right)$$

In the remainder, we omit edge features from formulas when unnecessary in favor of brevity.

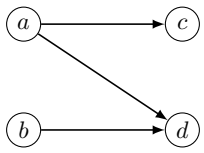


Figure 2: Nodes (a and b) with different out-degrees are not distinguishable by a standard MPNN with directed message passing. Note that naive bidirectional message passing, on the other hand, is unable to distinguish nodes a and d.

4 Methods

In this section, we introduce simple adaptations for standard MPNNs (Message Passing Neural Networks) to enable the detection of the fraud patterns in Fig. 1. We consider the adaptations in increasing order of complexity in terms of the patterns they help to detect. We provide theory results to motivate the adaptations and include corresponding experiments on the synthetic subgraph detection dataset in Section 7.1 to support the theory empirically.

4.1 Reverse Message Passing

When using a standard MPNN with directed edges, a node does not receive any messages from outgoing neighbors (unless they happen to also be incoming neighbors), and so is unable to count its outgoing edges. For example, a standard MPNN is unable to distinguish nodes a and b in Fig. 2. Further, note that naive bidirectional message passing, where edges are treated as undirected and messages travel in both directions, does not solve the problem, because a node then can not distinguish incoming and outgoing edges. So this would fail to distinguish nodes a and d in the same figure.

To overcome this issue, we need to indicate the direction of the edges in some way. We propose using a separate message-passing layer for the incoming and outgoing edges respectively, i.e., adding *reverse message passing*. Note that this is akin to using a relational GNN with two edge types (Schlichtkrull et al. 2018). More formally, the aggregation and update mechanisms become:

$$a_{in}^{(t)}(v) = \textsc{Agg}_{in}\left(\{h^{(t-1)}(u) \mid u \in N_{in}(v)\}\right),$$

$$a_{out}^{(t)}(v) = \textsc{Agg}_{out}\left(\{h^{(t-1)}(u) \mid u \in N_{out}(v)\}\right),$$

$$h^{(t)}(v) = \textsc{Update}\left(h^{(t-1)}(v), a_{in}^{(t)}(v), a_{out}^{(t)}(v)\right),$$

where a_{in} is now an aggregation of incoming neighbors and a_{out} of outgoing neighbors. We now prove that message-passing GNNs with reverse MP can solve degree-out.

Proposition 4.1. *An MPNN with sum aggregation and reverse MP can solve degree-out.*

The proof of Proposition 4.1 can be found in Appendix B. In Section 7.1, we use a synthetic pattern detection task to confirm that the theory translates into practice.

4.2 Directed Multigraph Port Numbering

People often make multiple transactions to the same account. In transaction networks, these are represented as parallel

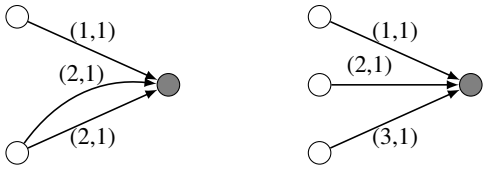


Figure 3: Nodes (in gray) with different fan-ins that are not distinguishable by a standard MPNN. The edge labels indicate incoming and outgoing port numbers, respectively.

edges. To detect fan-in (or fan-out) patterns, a model has to distinguish between edges from the same neighbor and edges from different neighbors. Using unique account numbers (or in general node IDs) would naturally allow for this. However, using account numbers does not generalize well. During training, a model can memorize fraudulent account numbers without learning to identify fraudulent patterns, but this will not generalize to unseen accounts.

Instead, we adapt port numbering (Sato, Yamada, and Kashima 2019) to directed multigraphs. Port numbering assigns local IDs to each neighbor at a node. This allows a node to identify messages coming from the same neighbor in consecutive message-passing rounds. To adapt port numbering to directed multigraphs, we assign each directed edge an incoming and an outgoing port number, and edges coming from (or going to) the same node, receive the same incoming (or outgoing) port number. Unlike Sato, Yamada, and Kashima (2019), who attach only the local port numbers at a node to received messages, we attach the port numbers in both directions, i.e., a node sees both the port number it has assigned to a neighbor and the port number that the neighbor has assigned to it. This turns out to be crucial for our expressivity arguments.

Port numbers have been shown to increase the expressivity of GNNs on simple graphs, but message-passing GNNs with port numbers alone cannot even detect 3-cycles in some cases (Garg, Jegelka, and Jaakkola 2020).

In general, the assignment of port numbers around a node is arbitrary. A node with d incoming neighbors can assign incoming port numbers in $d!$ ways. To break this symmetry in our datasets, we use the transaction timestamps to order the incoming (or outgoing) neighbors. In the case of parallel edges, we use the earliest timestamp to decide the order of the neighbors. Since timestamps carry meaning in financial crime detection, the choice of ordering is motivated; indeed two identical subgraph patterns with different timestamps can have different meanings.

Computing the port numbers in this way can be a time-intensive step, with runtime complexity dominated by sorting all edges by their timestamps: $\mathcal{O}(m \log m)$, where $m = |E(G)|$. However, port numbers can be calculated in advance, so training and inference times are unaffected. Each edge receives an incoming and an outgoing port number as additional edge features. Fig. 3 shows an example of graphs with port numbers. We now prove that GNNs using port numbers can correctly identify fan-in and fan-out patterns.

Note that the following proof, and later proofs using port numbers, do not rely on the timestamps for correctness. However, if timestamps that uniquely identify the ports are available, then permutation invariance/equivariance of the GNN will be preserved.

Proposition 4.2. *An MPNN with max aggregation and multigraph port numbering can solve fan-in.*

A proof is provided in Appendix B. Adding reverse MP, one can argue similarly that fan-out can also be solved. Both propositions are confirmed empirically in Section 7.1.

Proposition 4.3. *An MPNN with max aggregation, multigraph port numbering, and reverse MP can solve fan-out.*

4.3 Ego IDs

Although reverse MP and multigraph port numbering help with detecting some of the suspicious patterns in Fig. 1, they are not sufficient to detect directed cycles, scatter-gather patterns, and directed bicliques. You et al. (2021) introduced ego IDs specifically to help detect cycles in graphs. The idea is that by "marking" a "center" node with a distinct (binary) feature, this node can recognize when a sequence of messages cycles back around to it, thereby detecting cycles that it is part of. However, it turns out that the proof of Proposition 2 in the paper is incorrect, and ego IDs alone do not enable cycle detection. We give a counterexample in Fig. 6 in the appendix. Indeed, Huang et al. (2022) also note that the proof "confuses walks with paths".

We see this reflected in the individual results in Table 1. Although ego IDs offer a boost in detecting short cycles, they do not help the baseline (GIN) in detecting longer cycles. This can also be explained theoretically: Assuming a graph has no loops (edges from a node to itself), walks of length two and three that return to the start node are also cycles since there is no possibility to repeat intermediate nodes. Therefore Proposition 2 from You et al. (2021) applies in these cases and it is not surprising that GIN+EgoIDs can achieve impressive F1 scores for 2- and 3-cycle detection.

However, in combination with reverse MP and port numbering, ego IDs can detect cycles, scatter-gather patterns, and bipartite subgraphs, completing the list of suspicious patterns. In fact, it can be shown that a suitably powerful standard MPNN with these adaptations can distinguish any two non-isomorphic (sub-)graphs, and given a *consistent* use of port-numbering they will not mistakenly distinguish any two isomorphic (sub-)graphs. GNNs fulfilling these two properties are often referred to as *universal*. The crux of the proof is showing how the ego ID, port numbers, and reverse MP can be used to assign unique IDs to each node in the graph. Given unique node IDs, sufficiently powerful standard MPNNs are known to be universal (Loukas 2019; Abboud et al. 2020).

Theorem 4.4. *Ego IDs combined with port numbering and reverse MP can be used to assign unique node IDs in connected directed multigraphs.*

The idea of the proof is to show how a GNN can replicate a labeling algorithm that assigns unique IDs to each node in an ego node's neighborhood. The labeling algorithm as well as the full proof are provided in Appendix B. The universality of the adaptations follows from this theorem.

Corollary 4.4.1. *GIN with ego IDs, port numbering, and reverse MP can theoretically detect any directed subgraph pattern.*

The proof follows from Theorem 4.4 above and Corollary 3.1 from Loukas (2019). Similar statements can be made for simple undirected graphs. One can remove the reverse MP from the assumptions since this is only needed to make the proof work with directed edges.

Theorem 4.5. *Ego IDs and port numbering can be used to assign unique node IDs in connected undirected graphs.*

Corollary 4.5.1. *GIN with ego IDs and port numbering can theoretically detect any subgraph in undirected graphs.*

The ablation study in Table 1 of Section 7.1 again supports the theoretical analysis. The combination of the three adaptations achieves impressive scores for all subgraph patterns.

Note that passing the port numbers of both incident nodes of an edge is crucial for inferring unique node IDs. Fig. 5 in the appendix illustrates this with a simple example. In particular, port numbering, as introduced by Sato, Yamada, and Kashima (2019), is not sufficient.

4.4 Complexity & Runtime

We propose a set of adaptations, so the final model complexity will depend on the choice of base GNN. We describe the additional runtime costs incurred by the adaptations in Appendix C. All in all, the adaptations add a constant factor to the runtime complexity in addition to a one-off precomputation cost of $\mathcal{O}(m \log(m))$. The empirical runtimes on AML Small HI using GIN can be seen in Appendix F.5.

5 Datasets

Synthetic Pattern Detection Tasks. The AML subgraph patterns seen in Fig. 1 are used to create a controllable testbed of synthetic pattern detection tasks. The key design principle is to ensure that the desired subgraph patterns appear randomly, rather than being inserted post hoc into a graph. The problem with inserting patterns is that it skews the random distribution, and simple indicators (such as the degrees of nodes) can be enough to solve the task approximately. For example, consider the extreme case of generating a random k-regular graph and then inserting a pattern. Nodes belonging to the pattern could be identified by checking whether their degree exceeds k. Additionally, if only inserted patterns are labeled, then randomly occurring patterns will be overlooked.

To ensure that the desired subgraph patterns appear randomly, we introduce the *random circulant graph* generator. Details of the generator and pseudocode can be found in Appendix D.1. The pattern detection tasks include degree-in/out (number of in/out edges), fan-in/out (number of unique in/out neighbors), scatter-gather, directed biclique, and directed cycles of length up to six. Detailed descriptions can be found in Appendix D.2.

Anti-Money Laundering (AML). Given the strict privacy regulations around financial data, real-world datasets are not readily available. Instead, we use simulated money laundering data (Altman et al. 2023). The simulator behind these datasets generates a financial transaction network by modeling agents (banks, companies, and individuals) in a virtual world. The generator uses well-established laundering patterns to add realistic money laundering (illicit) transactions. We use two small and two medium-sized datasets, one of each with a higher illicit ratio (HI) and with a lower illicit ratio (LI). The dataset sizes and illicit ratios are provided in Table 4 in the appendix. We use a 60-20-20 temporal train-validation-test split, i.e., we split the transactions after ordering them by their timestamps. Details can be found in Appendix E.

Ethereum Phishing Detection (ETH). Since banks do not release their data, we turn to cryptocurrencies for a real-world dataset. We use an Ethereum transaction network published on Kaggle (Chen et al. 2019a), where some nodes are labeled as phishing accounts. We use a temporal train-validation-test split, but this time splitting the nodes. We use a 65-15-20 split because the illicit accounts are skewed towards the end of the dataset. More details and dataset statistics can be found in Appendix E.

Real-World Directed Graph Datasets. The theory results and the subgraph detection tasks demonstrate the general purpose potential of the architectural adaptations. However, testing our model on real-world benchmark datasets is important to further support these claims. For lack of established directed multi-graph benchmarks, we have taken three directed graph datasets, Chameleon, Squirrel (Pei et al. 2020), and Arxiv-Year (Hu et al. 2020), and compare our approach with the state-of-the-art model for these benchmarks (Rusch et al. 2022). As these datasets are not the focus of this paper, we leave the experimental details and results to the appendix; please see Appendix G.

6 Experimental Setup

Base GNNs and Baselines. GIN with edge features (Hu et al. 2019) is used as the main GNN base model with our adaptations added on top. GAT (Veličković et al. 2017) and PNA (Velickovic et al. 2019) are also used as base models, and we refer to their adapted versions as Multi-GAT and Multi-PNA, respectively. All three are also considered baselines. Additionally, GIN with ego IDs can be considered an ID-GNN (You et al. 2021) baseline, and GIN with port numbering can be considered a CPNGNN (Sato, Yamada, and Kashima 2019) baseline. Since AML is an edge classification problem, we also include a baseline using edge updates (Battaglia et al. 2018), denoted GIN+EU. This approach is similar to replacing edges with nodes and running a GNN on said *line graph*, which recently achieved state-of-the-art (SOTA) results in self-supervised money laundering detection (Cardoso, Saleiro, and Bizarro 2022). We also include R-GCN (Schlichtkrull et al. 2018) as a baseline. We do not focus on including a more expansive range of GNN baselines, for the simple reason that without (the proposed) adaptations, they are not equipped to deal with directed multigraphs. However, some additional results with "more expressive" GNNs can be found in Appendix H.1. As far as we are aware, there are no other GNNs that one could expect to achieve SOTA results on directed multigraphs.

We include a baseline representing the parallel line of work in financial crime detection that uses pre-calculated graph-based features (GFs) and tree-based classifiers to classify nodes or edges individually. We train XGBoost (Chen and Guestrin 2016) and LightGBM (Ke et al. 2017) models on the individual edges (or nodes) using the original raw features combined with additional graph-based features. This approach has produced SOTA results in financial applications (Weber et al. 2019; Lo, Layeghy, and Portmann 2022).

Given the size of the AML and ETH datasets, we use neighborhood sampling (Hamilton, Ying, and Leskovec 2017) for all GNN-based models. Further details of the experimental setup for the different datasets can be found in Appendix F.

Scoring. Since we have very imbalanced datasets, accuracy and other popular metrics are not suitable. Instead, we use the minority class F1 score. This aligns well with what banks and regulators use in real-world scenarios.

7 Results

7.1 Synthetic Pattern Detection Results

The synthetic pattern detection results can be seen in Table 1. The degree-out results reveal that the standard message-passing GNNs are unable to solve the degree-out task, achieving F1 scores below 44%. However, all the GNNs that are equipped with reverse MP score above 98%, thus supporting Proposition 4.1. The next column shows that port numbering is the critical adaptation for solving fan-in, though the F1 score is quite high even for the baseline GIN. On the other hand, for the fan-out task, the combination of reverse MP and port numbering is needed to score above 99%. Again, these results support Propositions 4.2 and 4.3. The ablation study of cumulative adaptations on top of GIN also supports Corollary 4.4.1: The combination of reverse MP, port numbering, and ego IDs, scores high on all of the subtasks, with only 6-cycle detection coming in below 90%. We see similar results when using other base GNN models, with Multi-PNA achieving the best overall results. Moreover, on the more complex tasks — directed cycle, scatter-gather, and biclique detection — the combination of the three is what leads to the first significant improvement in F1 scores. In the most extreme case, scatter-gather detection, the minority class F1 score jumps from 67.84% with only reverse MP and port numbers to 97.42% when ego IDs are added. No adaptation alone comes close to this score, so it is clear that the combination is needed. Similar jumps can be seen for directed 4-, 5-, and 6-cycle, and biclique detection. Increasing the dataset size and restricting the task to only the "complex" subtasks further increases the scores, with 6-cycle detection also reaching above 97%. More details can be found in Appendix H.2. Additional ablations can also be found in the appendix. In particular, we rerun the experiments using random unique node IDs as input features and see that node IDs are unable to replace port numbers and ego IDs in practice.

7.2 AML Results

The results for the AML datasets can be seen in Table 2. For AML Small HI, we see that our adaptations boost the minority class F1 score of GIN from 28.7% to 57.2%, a gain of almost 30%. The largest improvements are brought by reverse MP and port numbering, taking the F1 score from 28.7% to 56.9%, whilst ego IDs do not make much difference here. The results for the other AML datasets show a similar trend with overall gains of 14.2%, 14.0%, and 10.7% for GIN, again with diminishing returns as more adaptations are added. The two rows corresponding to port numbering — *GIN+Ports* and *+Ports* — indicate clear gains from using port numbering, both when used alone and on top of reverse MP. The support for ego IDs is less clear, with clear gains when used as an individual adaptation but no significant gains when added on top of reverse MP and port numbering. Note that only two GNN layers were used, so this conclusion could change as more layers are added and longer cycles can also be detected. The full set of adaptations was tested with three other base models, GIN+EU (GIN with edge updates), PNA, and PNA+EU. In each case, and across almost all AML datasets, we see clear gains from using the adaptations. These gains underline the effectiveness and versatility of the approach. We further note that Multi-PNA+EU outperforms all of the baselines on all of the AML datasets. This is particularly impressive when compared with the tree-based methods using graph-based features (XGBoost+GFs and LightGBM+GFs) since the hand-crafted features align perfectly with the illicit money laundering patterns used by the simulator. Moreover, these tree-based methods have been SOTA in previous financial applications (Weber et al. 2019; Lo, Layeghy, and Portmann 2022).

Recall scores for individual money laundering patterns can be found in Appendix I. It is worth noting that the majority of the illicit transactions that belong to money laundering patterns are identified, and the overall dataset scores are greatly influenced by the proportion of lone (not belonging to a money laundering pattern) illicit transactions in the datasets. Lone illicit transactions are very difficult to identify.

For training times and inference throughput rates of models based on GIN, please see Table 5 in the appendix. Notably, with all the adaptations, the inference rate of Multi-GIN still surpasses 18k transactions per second on a single GPU.

7.3 ETH Results

Finally, we test our adaptations on a real-world financial crime dataset — Ethereum phishing account classification. The results are provided in Table 2. Similar to the AML datasets, we see a consistent improvement in final scores as we add the adaptations. In total, the minority class F1 score jumps from 26.9% without adaptations to 42.9% with reverse MP, port numbering, and ego IDs. Again, the largest single improvement is due to the reverse MP. In this case, Multi-GIN does not outperform all of the baselines, but the adaptations also significantly boost PNA performance, and Multi-PNA and Multi-PNA+EU beat all the baselines by more than 12%.

8 Conclusion

This work has investigated a series of straightforward adaptations capable of transforming conventional message-passing GNNs into provably powerful directed multigraph learners. Our contributions to the field of graph neural networks are

Model	deg-in	deg-out	fan-in	fan-out	C2	C3	C4	C5	C6	S-G	B-C
GIN (Xu et al. 2018; Hu et al. 2019)	99.77	43.58	95.57	35.91	34.67	58.00	50.80	43.12	48.59	69.31	63.12
GAT (Veličković et al. 2017)	10.33	10.53	9.69	0.00	0.00	0.00	25.86	0.00	0.00	0.00	0.00
PNA (Velickovic et al. 2019)	99.63	43.02	95.00	38.93	25.77	54.75	51.92	48.79	48.40	65.88	65.51
GIN+EU (Battaglia et al. 2018)	99.30	42.74	95.70	39.13	32.58	55.91	54.65	47.62	49.68	68.54	64.64
GIN+EgoIDs (You et al. 2021)	99.78	51.48	95.06	49.24	98.13	97.97	53.12	44.37	45.42	66.44	63.90
GIN+Ports (Sato, Yamada, and Kashima 2019)	99.47	45.00	99.59	41.51	27.79	56.11	42.68	41.11	44.99	67.99	65.76
GIN+ReverseMP (Jaume et al. 2019)	98.87	99.08	94.99	95.25	35.96	63.85	69.09	67.44	71.23	65.83	66.18
+Ports	98.41	98.35	98.51	99.16	39.15	63.58	69.00	70.35	75.04	67.84	65.78
+EgoIDs (Multi-GIN)	99.48	99.09	99.62	99.32	98.97	98.73	97.46	91.60	84.23	97.42	94.33
Multi-GAT	98.68	98.36	99.28	99.33	98.61	98.93	98.90	95.82	91.81	96.66	86.92
Multi-PNA	99.64	99.25	99.53	99.41	99.71	99.54	99.49	97.46	88.75	99.07	96.77
Multi-GIN+EU	99.55	99.53	99.76	99.77	99.37	99.71	98.73	95.73	88.38	98.81	97.82

Table 1: Minority class F1 scores (%) for the synthetic subgraph detection tasks. First from the top are the standard MPNN baselines; then the results with each adaptation added separately on top of GIN; followed by GIN with the adaptations added cumulatively; and finally, results for the other GNN baselines with the three adaptations (Multi-GNNs). The Ck abbreviations stand for directed k-cycle detection, S-G stands for scatter-gather and B-C stands for biclique detection. We report minority class F1 scores averaged over five runs. We omit standard deviations in favor of readability.

Model	AML Small HI	AML Small LI	AML Medium HI	AML Medium LI	ETH
LightGBM+GFs (Altman et al. 2023)	62.86 ± 0.25	20.83 ± 1.50	59.48 ± 0.15	20.85 ± 0.38	53.20 ± 0.60
XGBoost+GFs (Altman et al. 2023)	63.23 ± 0.17	27.30 ± 0.33	65.70 ± 0.26	28.16 ± 0.14	49.40 ± 0.54
GIN (Xu et al. 2018; Hu et al. 2019)	28.70 ± 1.13	7.90 ± 2.78	42.20 ± 0.44	3.86 ± 3.62	26.92 ± 7.52
PNA (Velickovic et al. 2019)	56.77 ± 2.41	14.85 ± 1.46	59.71 ± 1.91	27.73 ± 1.65	51.49 ± 4.26
GIN+EU (Battaglia et al. 2018)	47.73 ± 7.86	20.62 ± 2.41	49.26 ± 4.02	6.19 ± 8.32	33.92 ± 7.34
R-GCN (Schlichtkrull et al. 2018)	41.78 ± 0.48	7.43 ± 0.38	OOM	OOM	OOM
GIN+EgoIDs (You et al. 2021)	39.65 ± 4.73	14.98 ± 2.66	45.26 ± 2.16	11.17 ± 6.41	26.01 ± 2.27
GIN+Ports (Sato, Yamada, and Kashima 2019)	54.85 ± 0.89	21.41 ± 2.40	54.22 ± 1.94	10.51 ± 12.82	32.96 ± 0.25
GIN+ReverseMP (Jaume et al. 2019)	46.79 ± 4.97	15.98 ± 4.39	51.93 ± 2.90	14.00 ± 9.34	36.86 ± 8.12
+Ports	56.85 ± 2.64	23.80 ± 4.07	57.15 ± 0.76	11.39 ± 8.36	42.51 ± 7.16
+EgoIDs (Multi-GIN)	57.15 ± 4.99	22.12 ± 2.88	56.23 ± 1.51	14.55 ± 2.91	42.86 ± 2.53
Multi-GIN+EU	64.79 ± 1.22	26.88 ± 6.63	58.92 ± 1.83	16.30 ± 4.73	48.37 ± 6.62
Multi-PNA	64.59 ± 3.60	30.65 ± 2.00	65.67 ± 2.66	33.23 ± 1.31	65.28 ± 2.89
Multi-PNA+EU	68.16 ± 2.65	33.07 ± 2.63	66.48 ± 1.63	36.07 ± 1.17	66.58 ± 1.60

Table 2: Minority class F1 scores (%) for the AML and ETH tasks. HI indicates a higher illicit ratio and LI indicates a lower illicit ratio. The models are organized as in Table 1. "OOM" indicates that the model ran out of GPU memory.

threefold. Firstly, our theoretical analysis addresses a notable gap in the existing literature about the power of combining different GNN adaptations/augmentations. Specifically, we prove that ego IDs combined with port numbering and reverse message passing enable a suitably powerful message-passing GNN, such as GIN, to compute unique node IDs and therefore detect any directed subgraph patterns. Secondly, our theoretical findings are validated empirically with a range of synthetic subgraph detection tasks. The practical results closely mirror the theoretical expectations, confirming that the combination of all three adaptations is needed to detect the more complex subgraphs. Lastly, we show how our adaptations can be applied to two important financial crime problems: detecting money laundering transactions and phishing accounts. GNNs enhanced with our proposed adaptations achieve impressive results in both tasks, either matching or surpassing relevant baselines. Reverse message passing and port numbering again prove crucial in reaching the highest scores, however, we find that ego IDs do not provide much additional benefit for these datasets.

Although this work has focused on financial crime applications, the theory and practical results have a broader relevance. Immediate future work could involve exploring applications of our methods to other directed multigraph problems. An initial exploration can be found in the appendix, showing promising results on three real-world datasets. However, further experiments are needed to confirm general applicability in various domains. Additionally, future work could explore the relationship between the computational complexity of different subgraph detection problems and GNN performance.

Acknowledgements

The support of the Swiss National Science Foundation (project numbers: 172610 and 212158) for this work is gratefully acknowledged.

References

Abboud, R.; Ceylan, I. I.; Grohe, M.; and Lukasiewicz, T. 2020. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*.

Altman, E.; Blanuša, J.; Von Niederhäusern, L.; Egressy, B.; Anghel, A.; and Atasu, K. 2023. Realistic Synthetic Financial Transactions for Anti-Money Laundering Models. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Babai, L.; Erdos, P.; and Selkow, S. M. 1980. Random graph isomorphism. *SIaM Journal on computing*, 9(3): 628–635.

Barceló, P.; Geerts, F.; Reutter, J.; and Ryschkov, M. 2021. Graph neural networks with local graph parameters. *Advances in Neural Information Processing Systems*, 34: 25280–25293.

Battaglia, P.; Pascanu, R.; Lai, M.; Jimenez Rezende, D.; et al. 2016. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29.

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Blanusa, J.; Ienne, P.; and Atasu, K. 2022. Scalable Fine-Grained Parallel Cycle Enumeration Algorithms. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, 247–258.

Blanuša, J.; Atasu, K.; and Ienne, P. 2023. Fast Parallel Algorithms for Enumeration of Simple, Temporal, and Hop-Constrained Cycles. ArXiv:2301.01068 [cs].

Bongini, P.; Bianchini, M.; and Scarselli, F. 2021. Molecular generative graph neural networks for drug discovery. *Neurocomputing*, 450: 242–252.

Bouritsas, G.; Frasca, F.; Zafeiriou, S. P.; and Bronstein, M. 2022. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Cardoso, M.; Saleiro, P.; and Bizarro, P. 2022. LaundroGraph: Self-Supervised Graph Representation Learning for Anti-Money Laundering. In *Proceedings of the Third ACM International Conference on AI in Finance*, 130–138.

Chen, L.; Peng, J.; Liu, Y.; Li, J.; Xie, F.; and Zheng, Z. 2019a. XBLOCK Blockchain Datasets: InPlusLab Ethereum Phishing Detection Datasets. http://xblock.pro/ethereum/.

Chen, T.; and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794.

Chen, Y.; Wei, Z.; and Huang, X. 2018. Incorporating corporation relationship via graph convolutional neural networks for stock price prediction. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 1655–1658.

Chen, Z.; Chen, L.; Villar, S.; and Bruna, J. 2020. Can graph neural networks count substructures? *Advances in neural information processing systems*, 33: 10383–10395.

Chen, Z.; Villar, S.; Chen, L.; and Bruna, J. 2019b. On the equivalence between graph isomorphism testing and function approximation with gnns. *Advances in neural information processing systems*, 32.

Derrow-Pinion, A.; She, J.; Wong, D.; Lange, O.; Hester, T.; Perez, L.; Nunkesser, M.; Lee, S.; Guo, X.; Wiltshire, B.; et al. 2021. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 3767–3776.

Dwivedi, V. P.; Luu, A. T.; Laurent, T.; Bengio, Y.; and Bresson, X. 2021. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*.

Egressy, B.; and Wattenhofer, R. 2022. Graph Neural Networks with Precomputed Node Features. *arXiv preprint arXiv:2206.00637*.

Feng, F.; He, X.; Wang, X.; Luo, C.; Liu, Y.; and Chua, T.-S. 2019. Temporal relational ranking for stock prediction. *ACM Transactions on Information Systems (TOIS)*, 37(2): 1–30.

Frasca, F.; Bevilacqua, B.; Bronstein, M. M.; and Maron, H. 2022. Understanding and extending subgraph gnns by rethinking their symmetries. *arXiv preprint arXiv:2206.11140*.

Garg, V.; Jegelka, S.; and Jaakkola, T. 2020. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning*, 3419–3430. PMLR.

Granados, O. M.; and Vargas, A. 2022. The geometry of suspicious money laundering activities in financial networks. *EPJ Data Science*, 11(1): 6.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.

He, J.; Tian, J.; Wu, Y.; Cia, X.; Zhang, K.; Guo, M.; Zheng, H.; Wu, J.; and Ji, Y. 2021. An efficient solution to detect common topologies in money launderings based on coupling and connection. *IEEE Intelligent Systems*, 36(1): 64–74.

Hornik, K.; Stinchcombe, M.; and White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5): 359–366.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33: 22118–22133.

Hu, W.; Liu, B.; Gomes, J.; Zitnik, M.; Liang, P.; Pande, V.; and Leskovec, J. 2019. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*.

Huang, Y.; Peng, X.; Ma, J.; and Zhang, M. 2022. Boosting the Cycle Counting Power of Graph Neural Networks with I^2-GNNs. *arXiv preprint arXiv:2210.13978*.

Jaume, G.; Nguyen, A.-p.; Martínez, M. R.; Thiran, J.-P.; and Gabrani, M. 2019. edGNN: a Simple and Powerful GNN for Directed Labeled Graphs. *arXiv preprint arXiv:1904.08745*.

Kanezashi, H.; Suzumura, T.; Liu, X.; and Hirofuchi, T. 2022. Ethereum Fraud Detection with Heterogeneous Graph Neural Networks. *arXiv preprint arXiv:2203.12363*.

Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; and Liu, T.-Y. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30.

Keisler, R. 2022. Forecasting global weather with graph neural networks. *arXiv preprint arXiv:2202.07575*.

Kipf, T. N.; and Welling, M. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

Li, C.; Jia, K.; Shen, D.; Shi, C.-J. R.; and Yang, H. 2019. Hierarchical Representation Learning for Bipartite Graphs. In *IJCAI*, volume 19, 2873–2879.

Li, W.; Bao, R.; Harimoto, K.; Chen, D.; Xu, J.; and Su, Q. 2021. Modeling the stock relation with graph network for overnight stock movement prediction. In *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence*, 4541–4547.

Liang, C.; Liu, Z.; Liu, B.; Zhou, J.; Li, X.; Yang, S.; and Qi, Y. 2019. Uncovering insurance fraud conspiracy with network learning. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, 1181–1184.

Lim, D.; Hohne, F.; Li, X.; Huang, S. L.; Gupta, V.; Bhalerao, O.; and Lim, S. N. 2021. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in Neural Information Processing Systems*, 34: 20887–20902.

Liu, Z.; Chen, C.; Yang, X.; Zhou, J.; Li, X.; and Song, L. 2018. Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM international conference on information and knowledge management*, 2077–2085.

Lo, W. W.; Layeghy, S.; and Portmann, M. 2022. Inspection-L: Practical GNN-based money laundering detection system for bitcoin. *arXiv preprint arXiv:2203.10465*.

Loukas, A. 2019. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199*.

Ma, Y.; Hao, J.; Yang, Y.; Li, H.; Jin, J.; and Chen, G. 2019. Spectral-based graph convolutional network for directed graphs. *arXiv preprint arXiv:1907.08990*.

Maron, H.; Ben-Hamu, H.; Serviansky, H.; and Lipman, Y. 2019. Provably powerful graph networks. *Advances in neural information processing systems*, 32.

Medvedev, E. 2022. ETH-etl. https://github.com/blockchain-etl/ethereum-etl. Accessed: 01-12-2022.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 4602–4609.

Nicholls, J.; Kuppa, A.; and Le-Khac, N.-A. 2021. Financial cybercrime: A comprehensive survey of deep learning approaches to tackle the evolving financial crime landscape. *Ieee Access*, 9: 163965–163986.

Papp, P. A.; Martinkus, K.; Faber, L.; and Wattenhofer, R. 2021. DropGNN: Random dropouts increase the expressiveness of graph neural networks. *Advances in Neural Information Processing Systems*, 34: 21997–22009.

Papp, P. A.; and Wattenhofer, R. 2022. A theoretical comparison of graph neural network extensions. In *International Conference on Machine Learning*, 17323–17345. PMLR.

Pei, H.; Wei, B.; Chang, K. C.-C.; Lei, Y.; and Yang, B. 2020. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*.

Rao, S. X.; Zhang, S.; Han, Z.; Zhang, Z.; Min, W.; Chen, Z.; Shan, Y.; Zhao, Y.; and Zhang, C. 2021. xFraud: explainable fraud transaction detection. *Proceedings of the VLDB Endowment*, 15: 427–436.

Rusch, T. K.; Chamberlain, B. P.; Mahoney, M. W.; Bronstein, M. M.; and Mishra, S. 2022. Gradient gating for deep multi-rate learning on graphs. *arXiv preprint arXiv:2210.00513*.

Sato, R.; Yamada, M.; and Kashima, H. 2019. Approximation ratios of graph neural networks for combinatorial problems. *Advances in Neural Information Processing Systems*, 32.

Sato, R.; Yamada, M.; and Kashima, H. 2021. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, 333–341. SIAM.

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Berg, R. v. d.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *Extended Semantic Web Conference*, 593–607. Springer.

Shu, K.; Wang, S.; and Liu, H. 2019. Beyond news contents: The role of social context for fake news detection. In *Proceedings of the twelfth ACM international conference on web search and data mining*, 312–320.

Starnini, M.; Tsourakakis, C. E.; Zamanipour, M.; Panisson, A.; Allasia, W.; Fornasiero, M.; Puma, L. L.; Ricci, V.; Ronchiadin, S.; Ugrinoska, A.; et al. 2021. Smurf-Based Anti-money Laundering in Time-Evolving Transaction Networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 171–186. Springer.

Suzumura, T. 2022. AMLSIM library wiki. https://github.com/IBM/AMLSim/wiki/Transaction-Model:-Alert-Model. Accessed: 30-11-2022.

Tong, Z.; Liang, Y.; Sun, C.; Li, X.; Rosenblum, D.; and Lim, A. 2020. Digraph inception convolutional networks. *Advances in neural information processing systems*, 33: 17907–17918.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Velickovic, P.; Fedus, W.; Hamilton, W. L.; Liò, P.; Bengio, Y.; and Hjelm, R. D. 2019. Deep Graph Infomax. *ICLR (Poster)*, 2(3): 4.

Watts, D. J.; and Strogatz, S. H. 1998. Collective dynamics of 'small-world'networks. *nature*, 393(6684): 440–442.

Weber, M.; Chen, J.; Suzumura, T.; Pareja, A.; Ma, T.; Kanezashi, H.; Kaler, T.; Leiserson, C. E.; and Schardl, T. B. 2018. Scalable graph learning for anti-money laundering: A first look. *arXiv preprint arXiv:1812.00076*.

Weber, M.; Domeniconi, G.; Chen, J.; Weidele, D. K. I.; Bellei, C.; Robinson, T.; and Leiserson, C. E. 2019. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591*.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Philip, S. Y. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1): 4–24.

Xu, B.; Shen, H.; Sun, B.; An, R.; Cao, Q.; and Cheng, X. 2021. Towards consumer loan fraud detection: Graph neural networks with role-constrained conditional random field. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 4537–4545.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Yang, S.; Zhang, Z.; Zhou, J.; Wang, Y.; Sun, W.; Zhong, X.; Fang, Y.; Yu, Q.; and Qi, Y. 2021. Financial risk analysis for SMEs with graph-based supply chain mining. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 4661–4667.

You, J.; Gomes-Selman, J. M.; Ying, R.; and Leskovec, J. 2021. Identity-aware graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, 10737–10745.

Zhang, M.; and Li, P. 2021. Nested graph neural networks. *Advances in Neural Information Processing Systems*, 34: 15734–15747.

Zhang, S.; Yao, L.; Sun, A.; and Tay, Y. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM computing surveys (CSUR)*, 52(1): 1–38.

Zhang, X.; He, Y.; Brugnone, N.; Perlmutter, M.; and Hirn, M. 2021. Magnet: A neural network for directed graphs. *Advances in neural information processing systems*, 34: 27003–27015.

Zhao, L.; Jin, W.; Akoglu, L.; and Shah, N. 2021. From stars to subgraphs: Uplifting any GNN with local structure awareness. *arXiv preprint arXiv:2110.03753*.

Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; and Sun, M. 2020. Graph neural networks: A review of methods and applications. *AI open*, 1: 57–81.

A Motivating Examples

A.1 Example Financial Transaction Graph

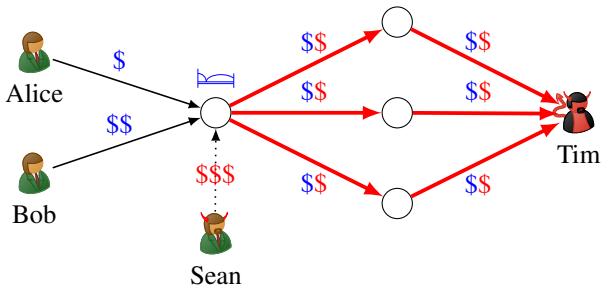


Figure 4: Example of money laundering in a network of financial transactions. Alice and Bob stay at a hotel, which is run by a criminal group headed by Tim. Sean has some dirty money (red dollars) from criminal activities that he wants to transfer to Tim. They use the hotel for laundering the money. They mix the dirty cash with clean money from guests, pay different contractors for supplies, and then transfer these payments to Tim. The money transfer from Sean to the hotel is a cash payment hidden from banks and financial authorities (dotted edge). However, the scatter-gather pattern (bold, red edges) could be revealing of a money laundering scheme.

A.2 Passing Port Numbers in Both Directions

Note that standard port numbering as introduced by Sato, Yamada, and Kashima (2019) is not sufficient for assigning unique node IDs. Unlike Sato, Yamada, and Kashima (2019) attach only the local port numbers at a node to received messages. In contrast, we attach the port numbers in both directions, i.e., a node sees both the port number it has assigned to a neighbor and the port number that the neighbor has assigned to it. This turns out to be crucial for inferring unique node IDs. For example, consider a simple undirected star graph with one internal node and 3 leaves, as shown in Fig. 5. The internal node assigns unique port numbers to the leaves, and according to (Sato, Yamada, and Kashima 2019), it concatenates each of these port numbers with the respective incoming messages from the three leaves, allowing it to distinguish the leaves and attribute the messages. However, the internal node's port numbers are not attached to outgoing messages, and since the leaves have a single port (neighbor), they receive identical messages with the identical port number attached in each message passing round. So the leaves have no way of attaining unique IDs when the internal node is the centre node with the ego ID. Similarly, the two unmarked nodes have no way of attaining unique node IDs when the third leaf is the centre node. Our approach solves this problem by attaching the port numbers in both directions to the messages.

A.3 Ego ID Cycle Detection Counterexample

We give a counterexample to show that ego IDs alone (You et al. 2021) are not sufficient to detect 6-cycles. Consider nodes u and u' in Fig. 6. They would receive the same label

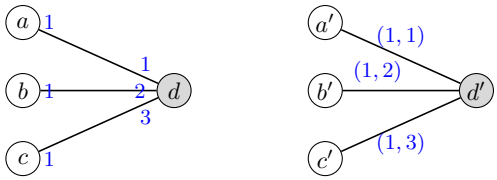


Figure 5: Sato, Yamada, and Kashima (2019) attach only the local port numbers to received messages, as indicated on the left. Port numbers are indicated in blue. For example, node c attaches port number 1 to all messages from d. So in particular, nodes a, b, and c can never be distinguished. We use the port numbers on either side on an edge as edge features, so both port numbers are seen by both incident nodes. In this example this might not be a problem — a, b, and c likely have the same ground truth labels — but this means that unique node IDs cannot be generated/propagated.

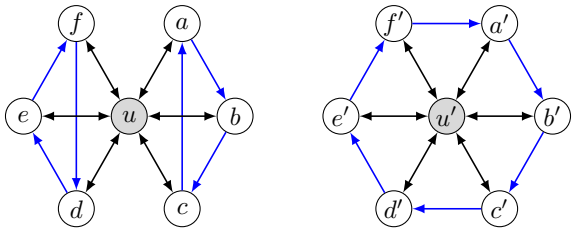


Figure 6: Example nodes (u and u') with different 5-cycle counts that are not distinguishable by a standard MPNN with ego IDs. For example, (u', a', b', c', d', u') is a 5-cycle on the right, but there are no 5-cycles involving u on the left. The example also works with undirected edges.

when using a standard MPNN with ego IDs (u and u' being the center nodes), but u' is part of several 5-cycles and u is not. To see intuitively why, note first that the blue edges form two 3-cycles on the one side and a 6-cycle on the other, and that the 1-WL test can not distinguish these alone. Now, when u and u' are the center nodes, the only extra information is that u and u' are marked, but since they are fully connected in their graphs, the mark does not help in distinguishing any of the other nodes, so the neighbors remain indistinguishable and indeed the two ego graphs remain indistinguishable. This means they could not both be labeled correctly in any task that relies on 5-cycles. Note that the predictions for any of the other respective nodes (e.g., a and a') will not be identical, since when these are the center nodes with the ego ID, then the GNN can detect that the ones on the left are in 3-cycles, whereas the ones on the right are not.

B Proofs

B.1 Proof of Proposition 4.1

Proof. Consider a message-passing GNN with sum aggregation. Then with a single GNN layer, initial node features $h^{(0)}(v) = 1$ for all $v \in V(G)$, we have $a_{in}^{(t)}(v) = d_{in}(v)$ and $a_{out}^{(t)}(v) = d_{out}(v)$. Choosing $\textsc{Update}\,(h, a_{in}, a_{out}) = a_{out}$, we get $h^{(1)}(v) = a_{out}^{(t)}(v) = d_{out}(v)$. □

B.2 Proof of Proposition 4.2

Proof. Since consecutive integers are used to number the incoming ports, the max aggregation function can directly output the maximum of the incoming port numbers to give the desired result. □

The statement also holds for MPNNs with sum aggregation. However, the proof is less intuitive and relies on the universality of sum aggregation for multisets (Lemma 5 from Xu et al. (2018)).

B.3 Proof of Theorem 4.4

For the purposes of this proof, we assume the Universal Approximation Theorem (Hornik, Stinchcombe, and White 1989) for multi-layer perceptrons (MLPs), stating that any continuous function with finite support can be approximated by an MLP with at least one hidden layer. We refer to this result to avoid giving explicit constructions for GNN layers. We also assume that the GNN uses min aggregation, consisting of an MLP applied elementwise to the set of incoming messages, followed by taking the minimum.

Proof. Initially, the ego node or *root* node starts with the node feature $h^{(0)}(r) = 1$, which will be its final node ID. The other nodes, $v \in V(G) \setminus \{r\}$ start with $h^{(0)}(v) = 0$, which indicates that they have not yet been assigned a final ID. The node IDs will be assigned layer by layer, such that a node that is k hops away from r receives a final ID after k rounds of message passing.

We call a node *active* if it has received a message from an active node in the previous round. At $t = 0$, only the root node is active. Once a node is active, it switches to inactive for all the remaining rounds. A GNN can keep track of active nodes, by using a dimension of the hidden node embeddings to indicate whether the node is currently active and another dimension to indicate whether the node has already been active. This information is then "included" in its messages, so neighboring nodes know when to activate. Notice that only nodes that are exactly distance k (where edges can be traversed in either direction) from the root node will be active in the k^{th} round.

By ignoring messages from inactive nodes, a GNN can replicate Algorithm 1. In each round, nodes receive messages from their neighbors, consisting of the send a message consisting of their ID and the incoming (or outgoing) port number to each of their incoming (or outgoing) neighbors respectively. The message is an integer in base $2n$ made by concatenating the node ID and the port number (n is added to the port number if it is an incoming edge). The message can be viewed as a node ID proposal, every active node proposes a node ID to each of its neighbors. Then all nodes that receive proposals accept the smallest of these proposals (again in base $2n$) as their node ID and become active.

In a GNN, this can be replicated from the receiver's perspective. In each round, nodes receive messages from their neighbors, consisting of the neighboring node's current state and the port numbers along the edge. The MLP at each node v, "ignores" messages from inactive nodes, e.g. by encoding them as a very high value. For the remaining messages, v encodes the neighbor's current state and port numbers as the concatenation of the neighbor's node ID (current state minus the two dimensions indicating activity) and the port number v has at its neighbor. In this way, the MLP essentially constructs the proposals from the algorithm. Node v then selects the minimum proposal.

Since our input graph is connected, every node in the graph will be reached (receive a proposal) within D rounds, since D is the diameter of the graph. Therefore every node will end up with a node ID.

It remains to be shown that this node ID will be unique. First, note that nodes at different distances from the root cannot end up with the same node ID. A node at distance k will receive its first proposal in round k and will therefore have an ID with exactly $k + 1$ digits. Finally, an inductive argument shows that active nodes (nodes at the same distance) cannot have the same node IDs. Certainly, this is true at the start when $X = \{r\}$. Now assuming all active nodes from the previous round $(k - 1)$ had distinct node IDs, then the only way two active nodes (in round k) can have the same ID is if they accept a proposal from the same neighboring node. This is because, assuming the induction hypothesis, proposals from different nodes will already differ in their first $k - 1$ digits. But if two active nodes accepted a proposal from the same node, then they would have received different port numbers — incoming ports are distinct, outgoing ports are distinct, and n is added to all incoming ports so that they cannot be the same as any outgoing ports. Therefore the active nodes always accept unique proposals. □

Algorithm 1: BFS node ID assignment (Theorem 4.4)

Input: Connected directed multigraph $G = (V, E)$ with n nodes, with diameter D and with root (or ego) node $r \in V$. Active nodes $X \subseteq V$ and finished nodes $F \subseteq V$. Port numbering $P : E \to \mathbb{N}^2$.

Output: Unique node IDs $h(v)$ for all $v \in V$ (in base $2n$)

$h(r) \leftarrow 1$; $h(v) \leftarrow 0$ for all $v \in V \setminus \{r\}$
$F \leftarrow \emptyset$; $X \leftarrow \{r\}$
for $k \leftarrow 1$ to D **do**
  for $v \in V$ **do**
    if $v \in X$ **then**
      send $h(v) \parallel P((v,u))_{out}$ to $u \in N_{out}(v)$
      send $h(v) \parallel n + P((u,v))_{in}$ to $u \in N_{in}(v)$
      $F \leftarrow F \cup \{v\}$; $X \leftarrow X \setminus \{v\}$
    if $v \notin F$ **then**
      if Incoming messages $M(v) \neq \emptyset$ **then**
        $h(v) \leftarrow \min\{M(v)\}$
        $X \leftarrow X \cup \{v\}$

The algorithm used in the proof is given in Algorithm 1. Fig. 7 gives an example output of the algorithm.

C Complexity & Runtime

We propose a set of adaptations, so the final time complexity of a model will depend on the choice of base GNN. We describe the additional runtime costs incurred by the adaptations below. The empirical runtimes using GIN can be seen

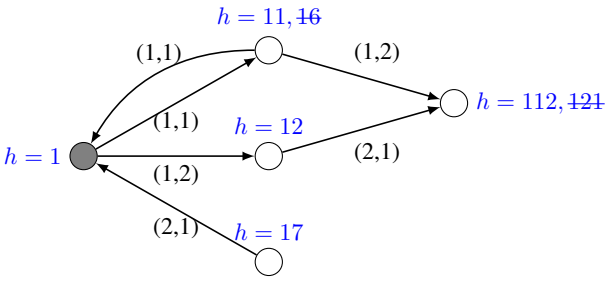


Figure 7: Example output of Algorithm 1 with the root (or ego) node indicated in gray. The edge labels show the (incoming, and outgoing) directed multigraph port numbers. Blue numbers indicate node ID proposals; declined proposals are struck through, leaving the final assigned node IDs. In this example $n = 5$ so node IDs are to be understood in base 10.

in Table 5, also in the appendix.

Reverse message passing multiples the time complexity by a constant factor. Theoretically, this constant factor would be 2, but in practice, we also increase the memory requirements of the model by a factor of 2, so the batch size has to be reduced, leading to a larger constant factor (see Table 5). Port numbering does not increase the time complexity of training and inference since it simply increases the size of the input edge features. This is confirmed by the empirical runtimes in Table 5. However, computing timestamp-based port numbers adds a pre-computation step of complexity $\mathcal{O}(m \log(m))$, which can be done in one go before training. Ego IDs similarly do not increase the runtime complexity, as can be seen in Table 5. All in all, the adaptations add a constant factor to the runtime complexity in addition to a pre-computation runtime of complexity $\mathcal{O}(m \log(m))$.

D Synthetic Pattern Detection Datasets

To ensure that the desired subgraph patterns appear randomly, we introduce the *random circulant graph* generator. The generator is parameterized by the number of nodes n, the average degree d, and a radius r controlling the local density of the generated graph. Together, these parameters control the rate of generating the various subgraph patterns. For example, by choosing a high average degree and a low radius, we can ensure that more dense patterns also appear randomly.

D.1 Random Circulant Graphs

A *random circulant graph*, $G_{n,d,r}$, has n nodes $\{1, 2, \ldots, n\}$, and $\lfloor \frac{nd}{2} \rfloor$ edges; one end, u, of each edge is sampled uniformly from $\{1, 2, \ldots, n\}$, and the other end, v, is sampled from a normal distribution around u with standard deviation r, and then rounded to the nearest integer, i.e., $v = \lfloor X + \frac{1}{2} \rfloor$, where $X \sim \mathcal{N}(u, r)$. Algorithm 2 shows a step-by-step description of the algorithm.

This model is similar to the Watts–Strogatz small-world model (Watts and Strogatz 1998), where nodes are also arranged in a ring, each node is connected to its k nearest neighbors, and edges are randomly rewired with a chosen probability. However, standard Watts–Strogatz implementations produce undirected graphs and do not allow for multiple edges between a pair of vertices. In particular, a randomly oriented Watts-Strogatz graph — a directed graph created by generating an undirected Watts-Strogatz graph, and then choosing an arbitrary direction for each edge independently and uniformly at random — doesn't allow for testing the detection of directed cycles of length 2 and fan patterns (without parallel edges fan patterns are identical to degree patterns).

Algorithm 2: Random Circulant Graph Generator

Parameters: Number of nodes n, average degree d, locality radius r.
Output: A graph $G = (V, E)$.

$N_{edges} \leftarrow \lfloor \frac{nd}{2} \rfloor$
$\vec{u} \leftarrow (u_i \mid i = 1, \ldots, N_{edges}) \overset{\text{iid}}{\sim} \text{unif}(\{1, \ldots, n\})$ ▷ Sample tail nodes
$\vec{v} \leftarrow (v_i \mid i = 1, \ldots, N_{edges}) \overset{\text{iid}}{\sim} \mathcal{N}(u_i, r)$ ▷ Sample head nodes
$\vec{v} \leftarrow (\lfloor v_i + \frac{1}{2} \rfloor \mid i = 1, \ldots, N_{edges})$ ▷ Round head nodes to nearest integer
$V(G) = \{1, \ldots, n\}$
$E(G) = \{(u_i, v_i) \mid i = 1, \ldots, N_{edges}\}$

D.2 Pattern Detection Tasks

The synthetic pattern detection dataset includes the following (sub-)tasks:

- **Degree-in (-out):** Degree -in and -out of a node is the number of incoming and outgoing transactions respectively. Note that multiple transactions can come from the same node, and these are counted separately.
- **Fan-in (-out):** Fan -in and -out of a node is the number of unique incoming and outgoing neighbors, respectively. Neighbors with multiple edges are counted only once, but a node can contribute to both the fan-in and fan-out counts, as it can be both an incoming and outgoing neighbor.
- **Cycle (k):** A node receives a positive label if it is part of a directed cycle of length k. We only consider directed cycle patterns and omit the word "directed".
- **Scatter-Gather:** A node receives a positive label if it is the *sink* node of a scatter-gather pattern with at least 2 *intermediate* nodes. A scatter-gather pattern consists of a *source* node, *intermediate* nodes, and a *sink* node. Given a *sink* node v, a node u is considered a *source* node if there is a directed path of length 2 from u to v. A node w_i is considered an *intermediate* node if there is a directed path of length 2 from u to v through w_i, i.e., if both (u, w_i) and (w_i, v) are in $E(G)$. Note that a scatter-gather pattern with a single intermediate node is simply a directed path of length 2. This does not capture the pattern we are interested in, so we require at least 2 intermediate nodes.
- **Gather-Scatter:** We do not explicitly include this pattern as a subtask since it is a combination of fan-in and fan-out.

Subtask	Threshold	Positive Ratio
Degree-in	>3	0.352
Degree-out	>3	0.349
Fan-in	>3	0.324
Fan-out	>3	0.323
Cycle (2)	-	0.191
Cycle (3)	-	0.344
Cycle (4)	-	0.527
Cycle (5)	-	0.677
Cycle (6)	-	0.779
Scatter-Gather	-	0.321
Biclique	-	0.318

Table 3: Synthetic pattern detection task statistics

- **Biclique:** A biclique (or complete bipartite graph) is a bipartite graph, where every node of the first set is connected to every node of the second set. We consider a special case of the directed biclique, where every node in the first set has at least one outgoing edge to every node in the second set. We call the nodes in the first set *source* nodes, and the nodes in the second set *sink* nodes. The task is to identify all nodes that are the sink nodes of some directed $K_{2,2}$ biclique with (at least) 2 source nodes and (at least) 2 sink nodes.

All the (sub-)tasks are combined in one overall task, i.e., the output of the models is a vector, where each element of the vector corresponds to one of the subtasks.

We implement pattern detection algorithms to find the desired patterns and generate binary node labels. The parameters are adjusted to achieve relatively balanced labels, but since a perfect split (for all subtasks) is not guaranteed, we rely on minority class F1 scores rather than accuracy to measure the performance of our models. All subtasks in the final dataset have at least 19% and at most 68% positive labels.

Thresholds are used in some cases to produce binary labels. For example, for the degree-in task, nodes with an in-degree greater than 3 receive the label 1, while low in-degree nodes receive the label 0. For degree-in/-out and fan-in/-out the threshold is 3. For degree-in-time-span and maximum-degree-in the thresholds are 1 and 4 respectively. Examples of the patterns are provided in Fig. 1.

We use $n = 8192$ nodes, average degree $d = 6$, and radius $r = 11.1$ to generate our pattern detection data. To avoid any information leakage between train, validation, and test sets, we generate an independent random circulant graph $G_{n,d,r}$ for each dataset. Table 3 shows the thresholds and distribution of labels for each subtask.

E AML and ETH Datasets

E.1 AML Data Split

We use a 60-20-20 temporal train-validation-test split, i.e., we split the transaction indices after ordering them by their timestamps. The data split is defined by two timestamps t_1 and t_2. Train indices correspond to transactions before time t_1, validation indices to transactions between times t_1 and t_2, and test indices to transactions after t_2. However, the validation and test set transactions need access to the previous transactions to identify patterns. So we construct train, validation, and test graphs. This corresponds to considering the financial transaction graph as a dynamic graph and taking three snapshots at times t_1, t_2, and $t_3 = t_{max}$. The train graph contains only the training transactions (and corresponding nodes). The validation graph contains the training and validation transactions, but only the validation indices are used for evaluation. The test graph contains all the transactions, but only the test indices are used for evaluation. This is the most likely scenario faced by banks and financial authorities who want to classify new batches of transactions.

Dataset	# nodes	# edges	Illicit Ratio
AML Small HI	0.5M	5M	0.07%
AML Small LI	0.7M	7M	0.05%
AML Medium HI	2.1M	32M	0.11%
AML Medium LI	2.0M	31M	0.05%
ETH	2.9M	13M	0.04%

Table 4: AML and ETH task statistics. HI indicates a higher illicit ratio and LI indicates a lower illicit ratio.

E.2 ETH Dataset

The Kaggle dataset was constructed by starting from 1165 reported phishing nodes and crawling the whole Ethereum transaction network via breadth-first search. The dataset originally comes with four edge features: transaction time, transaction amount, source node address, and destination node address. Each node has one feature, the account address. As this information is minimal, we enhance the dataset with additional transaction features taken directly from the blockchain using Ethereum ETL (Medvedev 2022). We add nonce, block number, gas, and gas price. These are cryptocurrency terms for the number of transactions made by the sender prior to the given transaction; the block containing this transaction; and the transaction fees the sender provided for the transaction to go through. Table 4 shows the number of nodes (accounts) and edges (transactions) and the illicit ratio. Unlike AML, this is a node classification task, where the goal is to classify nodes as phishing accounts.

Unlike the AML use case, we do not know a priori what subgraph patterns to look for to identify phishing nodes. To this end, we defined *fraudulent clusters* and visualized these to give an indication of the patterns our model might need to detect. The clusters were constructed by starting from the 1165 reported phishing nodes, adding their 1-hop neighborhoods, and then joining any clusters sharing a common node. Fig. 8 shows an example of one of these fraudulent clusters.

Similar to AML, we use a temporal train-validation-test split, but this time we split the nodes. We order the nodes by the first transaction they are involved in (either as sender or receiver) before splitting. Again this gives us threshold times t_1 and t_2, and we use these times to create our train, validation, and test graphs. See the AML dataset split for

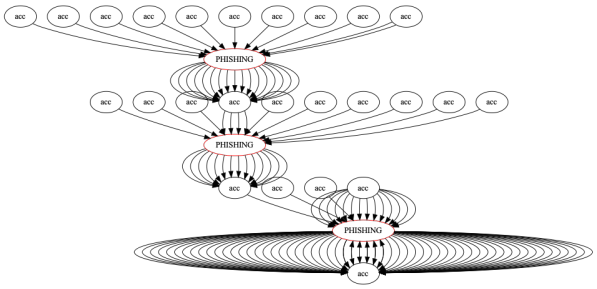


Figure 8: One of the fraudulent clusters identified in the ETH phishing dataset. The patterns that might help identify a fraudulent node are mostly 1-hop, namely in-degree, out-degree, fan-in, fan-out patterns, and 2-cycles.

Model	TTT (s)	TPS (inference)
GIN	1876	53130
Individual Adaptations		
GIN+Ports	2007	58663
GIN+EgoIDs	1873	58533
GIN+ReverseMP	12510	21067
Cumulative Adaptations		
GIN+ReverseMP+Ports	13051	18763
+EgoIDs	12958	18700

Table 5: Total training times (TTT) and Transaction per Second (TPS) for all GIN-based models on the AML Small HI dataset. The TPS values were measured in evaluation mode.

more details. This time we use a 65-15-20 split because the illicit accounts are skewed towards the end of the dataset, with few illicit accounts in the first half of the data.

F Experimental Details

F.1 Hyperparameter Tuning and Scoring

We used random sampling to identify a good range of hyperparameters. A second round of random sampling was conducted with this narrower range to pick our final set of hyperparameters. We varied the following hyperparameters: the number of GNN layers, hidden embedding size, learning rate, dropout, and minority class weight (for the weighted loss function). The number of random samples ranged between 10 to 50, depending on the training time of the model on a particular dataset.

To get our final results, we use the hyperparameters with the best validation score to train five models initialized with five different random seeds. The results of the five runs are then averaged and the standard deviation is calculated to give the reported scores.

F.2 LightGBM with Graph-based Features

We include a LightGBM baseline using pre-calculated graph features to classify nodes or edges individually. Calculating complex subgraph features naively can be prohibitively expensive, therefore we rely on efficient implementations (Blanuša, Atasu, and Ienne 2023; Blanusa, Ienne, and Atasu 2022; Altman et al. 2023) to enumerate subgraph patterns.

We use these graph-mining algorithms to enumerate the suspicious patterns introduced in Fig. 1. We enumerate on a fine-grained level, counting patterns of different sizes separately. For the edge classification tasks (AML) we count patterns that an edge is contained in and add these counts as additional edge features. We do the same with nodes for the node classification task (ETH).

We train a LightGBM (Ke et al. 2017) model on the individual edges (or nodes) with the full set of features (original raw features and additional graph-based features). The pattern enumeration can be restricted to a certain time range; we optimize the range (as a hyperparameter) for each dataset to get the best performance.

F.3 Synthetic Pattern Detection Experiments

Unlike the much larger AML and ETH datasets, we do not have to subsample the neighborhood. Instead, we use the whole 3-hop neighborhood in our model to ensure that all of the patterns can be found. Moreover, we used six GNN layers to ensure that 6-cycles can also be detected.

F.4 AML Experiments

Since this is an edge classification dataset, all GNNs use a final edge readout layer that takes the edge embedding and the corresponding endpoint node embeddings as its input. We use neighborhood sampling, sampling 100 one- and two-hop neighbors respectively. We remove node IDs from edge and node features to avoid, as far as possible, overfitting to the fraudulent node IDs. Since we know what financial crime patterns were used for generating this dataset, we use the graph mining library to enumerate exactly these suspicious patterns for each transaction: degree-in/-out, fan-in/-out, directed cycle, undirected cycle, and scatter-gather patterns of different sizes. We leave out bicliques, as these are not currently included in the library. For the low IR dataset, we took the hyperparameters from the corresponding high IR dataset and only optimized the minority class weight. We found that this has to be optimized to account for the greater class imbalance.

F.5 AML Runtimes

Table 5 shows the runtime cost of adding the adaptations to GIN when running on the AML Small HI dataset. The size of the GNN models was kept the same: 2 GNN layers and a hidden size of 64. All models were run on an Nvidia GeForce RTX 3090 GPU.

F.6 ETH Experiments

Since this is a node classification dataset, all GNNs use a final node readout layer that takes only the corresponding node embedding as its input. We use neighborhood sampling, sampling 1000 one- and two-hop neighbors respectively.

Dataset	# nodes	# edges	# node feats	# classes
Chameleon	2,277	36K	2,325	5
Squirrel	5,201	217K	2,089	5
Arxiv-Year	169K	1.17M	128	40

Table 6: Real-World dataset statistics.

Model	Squirrel	Chameleon	Arxiv-Year
Grad. Gating (SOTA)	$\mathbf{64.26 \pm 2.38}$	71.40 ± 2.38	63.30 ± 1.84
GIN (GCN*)	$35.94 \pm 2.01^*$	51.29 ± 0.63	50.15 ± 0.26
+ReverseMP	$59.84 \pm 2.08^*$	66.36 ± 1.23	63.31 ± 0.45
+Ports	$59.33 \pm 2.40^*$	73.46 ± 0.97	$\mathbf{68.12 \pm 0.51}$
+EgoIDs	$58.41 \pm 2.49^*$	$\mathbf{73.73 \pm 1.03}$	68.04 ± 0.60

Table 7: Comparison with state-of-the-art (Rusch et al. 2022) results on three real-world directed graph datasets.

For the LightGBM baseline, we use the graph mining library to enumerate the same graph-based patterns as for AML, but this time for nodes. Additionally, we calculate basic node statistics (such as averages, maximums, and variances) of the transaction amounts and the time between transactions to improve the baseline further. The statistical features give a significant boost to the minority class F1 scores, from 27.1% with only raw and graph-based features to above 50% with the additional statistical features.

G Additional Real-World Benchmarks

The theory results and the subgraph detection tasks demonstrate the general purpose potential of the architectural adaptations. Indeed, many of the subgraph patterns we consider could also be relevant to other areas. However, testing our model on real-world benchmark datasets is crucial to support these claims. Therefore we have taken three real-world directed graph datasets and compared our approach with the state-of-the-art (SOTA) model for these 3 benchmarks (Rusch et al. 2022). A summary of the dataset statistics can be seen in Table 6.

We report results for GIN with various adaptations. The models are set up as in our previous experiments. We use the Adam optimizer and train the model for 2000 epochs, using early stopping on the validation accuracy with a patience of 400. We do a single run of random sampling to optimize the hyperparameters within a predefined range. We sample 20 different sets of hyperparameters, and take choose the best performing model. We use the standard data splits for the datasets: For Chameleon and Squirrel we use the fixed GEOM-GCN splits (Pei et al. 2020), and for Arxiv-Year we use the splits provided by Lim et al. (2021). We report the mean and standard deviation of the test accuracy, calculated over 10 different data splits. The results can be seen in Table 7. We outperform SOTA in 2 out of the 3 benchmarks. On one dataset in particular the gain is almost 6 percentage points.

Model	C4	C5	C6	S-G	B-C
DropGIN	16.92	26.02	45.17	48.87	51.59
R-GCN	65.57	64.35	71.27	59.23	55.61
PPGN	57.93	48.74	41.15	54.23	54.98
Multi-DropGIN	99.09	97.08	94.93	96.98	94.27
Multi-R-GCN	85.20	74.08	74.30	82.37	67.16
Multi-PPGN*	98.11	92.49	86.69	68.74	62.78
Multi-PNA	99.49	97.46	88.75	99.07	96.77

Table 8: Additional subgraph detection results (F1 scores). Multi-PNA is included for comparison. *No port numbers.

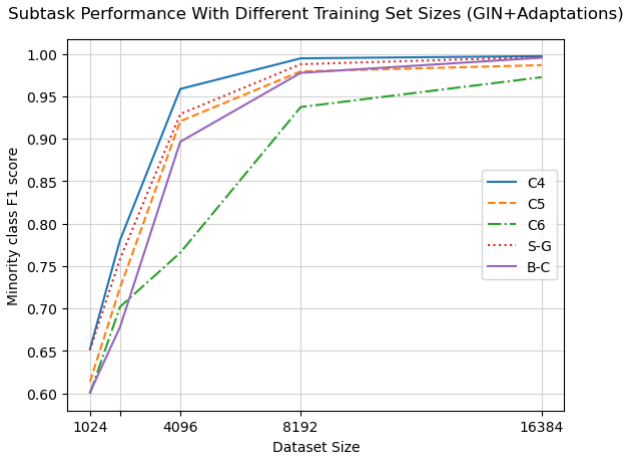


Figure 9: Performance of Multi-GIN on the "complex" subtasks as training dataset size is increased.

H Additional Subgraph Detection Results

H.1 Additional GNN Baselines

There are many "expressive" GNN architectures that have been developed for simple (no loops or parallel edges), undirected graphs with subgraph detection in mind. For example, PPGN (Maron et al. 2019) and DropGIN (Papp et al. 2021) to name a few. However, as we argue in Section 6 *Base GNNs and Baselines*, none of these has been developed for directed multigraphs, and their ability to detect subgraphs unfortunately does not translate into this setting. For instance, since none of these models consider the direction of edges, they cannot distinguish between incoming and outgoing neighbors. Some of these architectures could be adapted to the setting, and in some cases, our proposed adaptations offer a possible approach to do so. To support our argument, we have added more baselines for the subgraph detection experiments, please see Table 8 for the results. When possible, we also include adapted versions based on our proposed adaptations. The architectures are indeed unable to perform well without adaptation.

H.2 Complex Patterns and Dataset Size

Additional experiments were carried out to measure the influence of dataset size on the subgraph detection tasks. We restrict these experiments to the GIN model with all the adaptations. We also restrict to the more "complex" tasks that

No. Training Nodes	C4	C5	C6	S-G	B-C
1024	65.22 ± 3.34	61.38 ± 7.25	59.97 ± 7.30	65.12 ± 2.83	60.11 ± 2.86
2048	78.06 ± 6.19	72.44 ± 4.48	70.19 ± 2.49	75.82 ± 4.77	67.82 ± 1.53
4096	95.85 ± 0.86	92.06 ± 1.81	76.57 ± 0.96	92.90 ± 1.29	89.64 ± 1.74
8192*	97.46 ± 0.43	91.60 ± 1.50	84.23 ± 1.12	97.42 ± 0.85	94.33 ± 0.95
8192	99.46 ± 0.05	97.92 ± 0.26	93.72 ± 0.75	98.77 ± 0.18	97.75 ± 0.21
16384	99.72 ± 0.06	98.66 ± 0.15	97.24 ± 0.19	99.61 ± 0.08	99.53 ± 0.07

Table 9: Minority class F1 scores for the "complex" subgraph detection tasks with different dataset sizes. Multi-GIN is used in all experiments. The * indicates that these results are copied from Table 1, where all subgraph tasks were learned at once. All other experiments were restricted to the subtasks shown.

No. Training Nodes	C2	C3	C4	C5	C6	C7	C8	C9	C10
2048	97.53	89.72	84.71	80.60	75.48	81.34	91.51	93.57	93.39
4096	99.51	96.49	95.23	89.75	85.71	85.85	90.69	94.52	96.09
8192	99.83	99.78	99.13	97.17	93.58	89.93	91.87	94.92	96.64
16384	99.96	99.95	99.75	99.01	97.46	95.89	95.15	96.45	97.64
32768	99.99	100.00	99.93	99.62	98.76	97.98	97.93	98.10	98.30
65536	100.00	100.00	99.96	99.80	99.36	98.89	98.73	98.86	98.83

Table 10: Minority class F1 scores for detecting cycles of increasing lengths with different training dataset sizes. Multi-GIN is used in all experiments.

Model	deg-in	deg-out	fan-in	fan-out	C2	C3	C4	C5	C6	S-G	B-C
GIN	99.63	45.70	99.49	42.15	35.48	57.04	51.07	45.66	48.32	69.38	63.85
GIN+EgoIDs	99.90	51.51	99.64	49.92	98.89	93.71	57.89	49.08	49.78	67.27	62.88
GIN+Ports	99.84	42.33	99.51	38.61	33.72	55.77	55.43	47.25	46.79	68.89	65.68
GIN+ReverseMP	98.64	98.73	97.74	98.83	40.09	61.10	72.34	70.27	74.55	66.44	66.26
+Ports	99.03	99.44	99.35	99.37	43.15	63.15	68.20	68.29	73.67	68.02	66.87
+EgoIDs (Multi-GIN)	99.68	99.71	99.76	99.52	99.02	99.13	97.93	93.88	82.83	99.47	98.81

Table 11: Subgraph detection results with parallel edges combined into single weighted edges.

Model	deg-in	deg-out	fan-in	fan-out	C2	C3	C4	C5	C6	S-G	B-C
GIN	99.91	44.68	95.26	40.72	25.05	55.30	46.73	43.61	47.32	65.97	63.58
GIN+EgoIDs	99.82	52.56	95.67	49.78	98.67	95.78	54.56	45.95	48.01	69.00	64.93
GIN+Ports	98.88	44.49	99.16	40.71	27.84	55.73	48.03	43.30	47.78	66.46	63.04
GIN+ReverseMP	98.58	99.24	93.19	94.23	38.18	62.62	69.74	69.08	74.07	65.83	63.67
+Ports	98.36	98.31	99.01	99.37	37.13	61.88	72.12	71.04	76.47	65.00	66.89
+EgoIDs (Multi-GIN)	99.31	99.05	99.36	99.37	98.28	97.97	95.92	87.82	78.15	96.83	91.26

Table 12: Subgraph detection results using random unique integer node IDs.

Model	Fan-in	Fan-out	Cycle	S-G	G-S	B-C	none
GIN (Xu et al. 2018; Hu et al. 2019)	40.65	64.89	44.44	51.46	47.24	34.85	4.31
GIN+ReverseMP (Jaume et al. 2019)	58.54	80.92	40.40	72.80	68.50	36.36	2.43
+Ports	67.48	91.60	48.48	82.85	84.51	51.52	1.08
+EgoIDs (Multi-GIN)	87.80	80.15	50.51	89.96	86.35	51.52	1.89
Multi-GIN+EU	88.62	95.42	57.58	88.28	94.49	57.58	1.35
Multi-PNA	85.37	93.13	61.62	93.72	92.65	54.55	0.54

Table 13: Recall scores by money laundering pattern on AML Small HI dataset, where ground truth pattern labels are available. S-G stands for Scatter-Gather, G-S stands for Gather-Scatter, B-C stands for Bipartite, and none indicated illicit transactions that are not part of any standard money laundering pattern. The ground truth money laundering patterns are of varying sizes.

Model	Fan-in	Fan-out	Cycle	S-G	G-S	B-C	none
GIN (Xu et al. 2018; Hu et al. 2019)	0.00	60.87	11.90	20.34	41.43	2.38	1.45
GIN+ReverseMP (Jaume et al. 2019)	14.29	36.96	26.19	37.29	34.29	28.57	5.58
+Ports	35.71	76.09	42.86	54.24	54.29	42.86	3.31
+EgoIDs (Multi-GIN)	42.86	56.52	28.57	45.76	51.43	40.48	0.83
Multi-GIN+EU	42.86	84.78	40.48	52.54	70.00	35.71	0.83
Multi-PNA	71.43	95.65	33.33	71.19	94.29	64.29	0.62

Table 14: Recall scores by money laundering pattern on AML Small LI dataset, where ground truth pattern labels are available.

were not solved almost perfectly, namely C4, C5, C6, Scatter-Gather, and Biclique detection. Dataset sizes of 1024 up to 16384 are used, increasing by a factor of 2. All other graph generator settings are kept constant. In particular, we use the same average degree and average radius.

The results can be seen in Fig. 9, and specific values can be found in Table 9. One can see that increasing the training dataset size gradually increases the scores on all the complex tasks, with even 6-cycle detection reaching above 97% minority class F1 score. The starred results also show that restricting to a smaller set of tasks significantly improves performance.

For cycles, we further extended these experiments to cycles of length 10 and training datasets with 65536 nodes. These results can be seen in Table 10 and show similar trends with all cycle sizes reaching above 98% minority class F1 score with the largest training dataset.

H.3 Subgraph Detection with Unique Node IDs

We include additional subgraph detection results below. Firstly, we rerun the subgraph detection experiments with all GIN-based models with the addition of random unique node IDs as part of the input. The results can be seen in Table 12. This experiment shows that theoretical expressiveness does not always translate into better results in practice. Comparing the first row of the table with the first row of Table 1 reveals that random node IDs, although universal, are not effective for subgraph detection. The rest of the table shows the effect of adding adaptations. The trends are the same, and the results are similar to Table 1, but most of the scores are actually slightly lower with the node IDs. Note in particular that both port numbers and ego IDs (in addition to reverse message passing) are still needed to attain high scores for the complex patterns. This highlights that although node IDs could easily provide either of these features, the neural network is not able to extract and use this information effectively. The results indicate that the adaptations offer the right inductive biases for the task.

H.4 Subgraph Detection with the Collapsed Multigraph

Secondly, we explore whether the directed multigraph input can be collapsed into a directed graph with weighted edges, where the edge weights indicate the number of parallel edges in the original multigraph. We rerun the subgraph detection experiments with all GIN-based models. The results of the collapsed multigraph experiments can be found in Table 11. The results are very similar to using the But importantly, note that port numbers are no longer needed to solve the tasks. For example, fan-in can now be solved perfectly without ports, and port numbers do not increase any of the scores significantly, whether added on top of GIN or GIN with reverse message passing. This is not surprising as all of these tasks can be solved easily on a weighted directed graph (given reverse message passing and ego IDs as needed). Indeed, collapsing the parallel edges of a multigraph in this way could be a valid option for some multigraph tasks. However, as soon as edges have non-additive features that are important for the task, this might no longer be an option. It might not be possible to summarize the features of parallel edges without losing valuable information.

I Detailed AML Results

Ground truth money laundering pattern labels are available for a portion of the fraudulent transactions in the AML datasets. The remaining illicit transactions (without pattern

labels) do not belong to specific money laundering patterns; we might therefore expect worse recall scores for these transactions. Using these pattern labels, we calculate recall scores for each pattern type individually and present these results in Table 13 for AML Small HI and in Table 14 for AML Small LI.

From the tables, we can see that our models identify most of the laundering patterns leading to very high individual recall scores, though the cycle and bipartite recall scores could be further improved. This is likely due to the fact that our best-performing GNN models used only two message passing layers, making it impossible to detect longer cycles. Increasing the number of layers can increase the overall F1 scores, but also comes at a significant cost in terms of the runtime.

We can also immediately notice that the recall scores for illicit transactions that do not belong to any specific money laundering pattern (none) are close to 0%. This probably explains the low aggregate minority class F1 scores for Small LI (Table 2) where 71% of the laundering transactions are not part of laundering patterns. The equivalent value for Small HI is only 38%. These insights reveal just how effective the proposed approach is in detecting money laundering patterns.