ANTI-MONEY LAUNDERING IN BITCOIN BY USING DEEP GRAPH INFOMAX

September 16, 2021

1 Introduction

The advent of the first cryptocurrency - Bitcoin [1], has revolutionised the conventional financial ecosystem by enabling low-cost, anonymous, peer-to-peer cash transfers within and across various borders. However, its pseudonymity allows many cybercriminals, terrorists, and hackers to it them for illegal transactions. For example, the well-known computer virus WannaCry would ask victims to pay the ransom using Bitcoin after infecting one computer due to Bitcoin's non-traceability [2]. According to Elliptic, criminals received about 3.4 million (46.4 BTC) ransom only four days after WannaCry started to spread [2]. Despite their negative impact, cryptocurrencies are gold mines for open-source intelligence as transaction network data are publicly available, allowing law enforcement agencies to conduct forensic analysis on currency flows.

Therefore, detecting illicit nodes in Blockchain transaction graphs is extremely important in combating illegal transactions. This problem is challenging for law enforcement agencies because of the untraceable nature of Bitcoin transactions. Fortunately, however, graph representation learning algorithms have shown great potential for detecting illegal activities involving cryptocurrencies. In this paper, we will apply a graph neural network-based algorithm to cluster (or categorise) licit and illicit nodes in a real-world Blockchain transaction graph dataset published by Elliptic [3]. This Bitcoin temporal graph-based transaction dataset is partially labelled and consists of real-world entities belonging to licit categories (e.g., wallet, miners), illicit entities (e.g., scams, terrorist organisations, ransomware), and unknown categories.

Formally, we formulate the problem of node clustering as follows: given a transaction graph $G(t) = (V(t), E(t))$ in which $V(t)$ is the set of graph nodes and $E(t)$ is the set of graph edges with a timestamp t, we need to cluster the nodes $V(t)$ into the three categories mentioned above.

There are several critical challenges to this task. Unlike typical graph datasets, the Elliptic graph dataset is a temporal graph-based dataset consisting of multiple $G(t)$ graphs, and the current graph representation learning methods rarely focus on temporal graphs. Moreover, training models on this dataset also suffers from the problem of imbalanced class, in the sense that there are only a few illicit nodes in all any graph $G(t)$. This problem might significantly affect the clustering performance [4] and may result in the algorithm classifying every node in the test set as a licit node. In this paper, we propose GraphSAGE [5] with Deep Graph Infomax [6] to perform node clustering. We will investigate the temporal and class imbalance problems through experimental studies.

2 Related Work

Mark et al. [3] from Elliptic created and published the Elliptic dataset, a temporal graph-based Bitcoin transaction dataset consisting of over 200KB Bitcoin transaction nodes, 234K payment edges, and 49 graphs with timestamps as presented in Figure 1. Each of the transaction nodes has been labelled as either a "licit", "illicit", or "unknown" entity. They attempted to categorise the nodes in the Elliptic dataset using various machine learning methods, including Logistic Regression (LR), Random Forest (RF), Multilayer Perceptrons (MLP), Graph Convolutional Networks (GCN). They achieved a recall score of 0.67 with RF and 0.51 with GCN to retrieve illicit nodes.

Yining et al. [7] collected the Blockchain transaction graph data between July 2014 and May 2017 by running a Bitcoin client to collect the data and used an external trusted source "Wallet Explorer" [8], a website that tracks Bitcoin wallets,

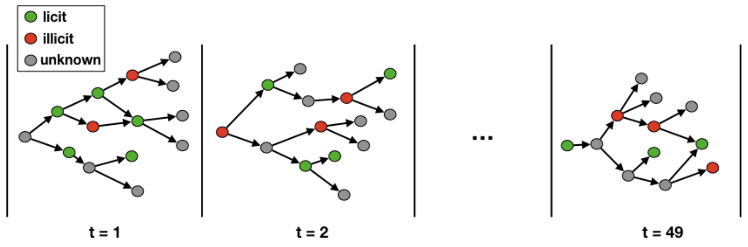


Figure 1: Structure of the dataset

to label the data. They highlighted the differences between money laundering and regular transactions, then applied a node2vec-based classifier to classify these two types of transactions. The research indicated that some properties of the transaction graph, including in-degree/out-degree, number of weakly-connected components and sum/mean/standard deviation of the output values, could distinguish money laundering transactions from legitimate ones. This classifier achieved an average accuracy score of 99.29% and an F1 score of 93%.

Chaehyeon et al. [9] applied supervised machine learning algorithms, i.e., Random Forest (RF) and Artificial Neural Network (ANN), to classify illicit nodes in the Bitcoin network. They collected legal and illegal Bitcoin data from forum sites "Wallet Explorer" [8] and "Blockchain Explorer" [10]. After that, they extracted features from the collected data. The resulting features were then fed to the RF and ANN. In their experiments, ANN and RF achieved 0.89 and 0.98 F1 scores, respectively.

Alarab et al. [11] proposed using an ensemble model consisting of a Random Forest, Extra Trees, and Bagging classifiers to classify illicit nodes [3]. The overall classification accuracy and recall score achieved are 98.13% and 72%, respectively. The results indicated that the ensemble learning model outperformed all the methods mentioned in the original paper [3].

3 Background

3.1 Graph Neural Networks (GNN)

Graph neural networks (GNNs) have been widely applied to analyse data extracted from social media networks, telecommunication networks, online misinformation, etc. Similarly, they can also be adopted for data extracted from a Blockchain transaction network. The latent features of Bitcoin transaction graphs are helpful for anomaly detection. Unfortunately, current anomaly detection methods such as Random Forest may not achieve satisfactory performance in capturing graph topological patterns. This is because a large number of graph nodes can exist and interconnect with each other. Moreover, Convolutional Neural Networks (CNNs) cannot capture the topological patterns of graphs as their nodes have no particular order or spatial of locality [12]. Therefore, GNNs have been proposed to address these problems.

Graphs are an example of non-Euclidean data. GNNs apply message propagation to capture the topological patterns, as shown in the figure 2. The aggregator aggregates features of the k-hop neighbours of a graph node to compute node embeddings. This process will be repeated for several iterations to aggregate information from the node neighbours. Finally, the latent representation of each node can be computed. However, there are some limitations of this approach as traditional GNNs use full k-hop neighbours, and this practice will be memory-insufficient if the number of neighbours is extremely large [5].

As a result, *Graph SAmple and aggreGatE (GraphSAGE)* [5] have been proposed to address the aforementioned limitations. For the GraphSAGE algorithm, at each of the k-hop layers, the size of each node's neighbour should be specified before using uniform sampling - a neighbour sampling technique, to compute node embeddings. This technique can achieve fixed per-batch space and time complexity, and it is more efficient than the original model that does not use uniform sampling.

A key hyperparameter of the GraphSAGE algorithm is the number of sampled neighbours at the *graph convolutional layers* K, which specifies the number of k-hop neighbours to be aggregated.

Another important hyperparameter of GraphSAGE is the choice of the differentiable aggregator function AGG_k to aggregate the features of the k-hops neighbours. Different aggregators can be chosen, including mean, pooling, or

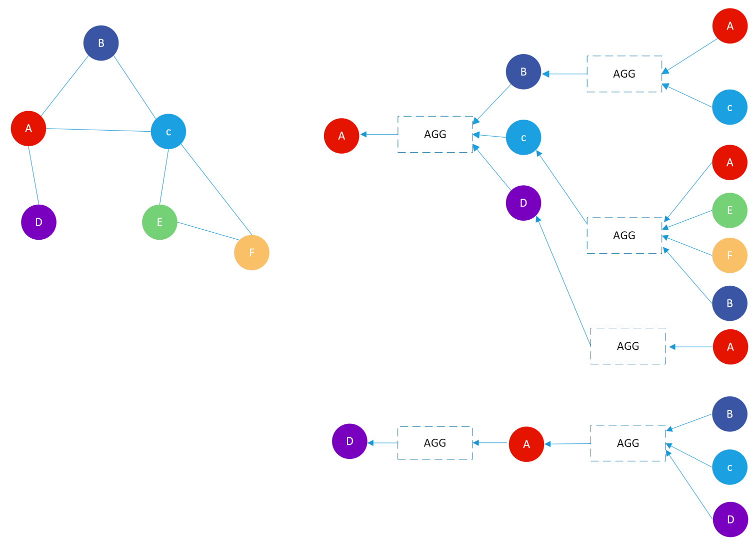


Figure 2: Overview of Graph Neural Network with 2-hops Layers

Long Short-Term Memory (LSTM) [5], etc. The aggregated information of the sampled neighbourhood $\mathbf{h}^k_{N(v)}$ is then concatenated with the node's embeddings from the previous layer $\mathbf{h}^{k-1}_v$.

Then, we perform dot product between the concatenated node's embeddings and trainable parameters $\mathbf{W}^k$. The resulting product is then passed through to a non-linear activation function σ (e.g., ReLU) to calculate node v embeddings v , as shown in Equation 1.

$$\mathbf{h}^k_v = \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}\left(\mathbf{h}^{k-1}_v, \mathbf{h}^k_{\mathcal{N}(v)}\right)\right) \tag{1}$$

3.2 Deep Graph Infomax

Deep Graph Infomax (DGI) [6] is an unsupervised graph representation learning approach. DGI relies on maximising the mutual information between patch representations and global graph summaries. The patch representations learnt summarise sub-graphs, allowing for the preservation of similarities on the patch level. Thus, the trained encoder in DGI can be reused to generate node embeddings for downstream tasks, such as node clustering. Most of the previous works on unsupervised learning with GCNs rely only on the random walk strategy, which is computationally expensive because the number of walks depends on the number of nodes on the graph. Hence, they are unsuitable for large graphs. In addition, the hyperparameter *length of the walk* can significantly impact the model performance. However, DGI does not require a label or random walk; instead, it guides the model to learn how the graph nodes are connected by randomly shuffling the node features of the graph. Therefore, it can be used to train node embeddings in an unsupervised manner.

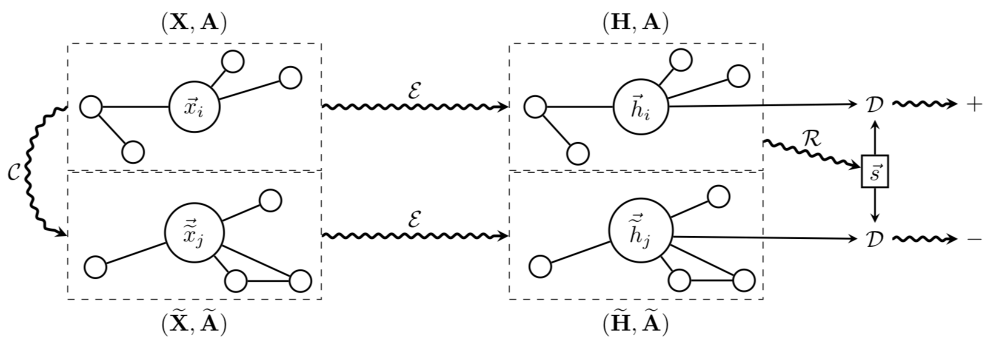


Figure 3: Overview of Deep Graph Infomax [6]

Figure 3 shows the overall procedure of DGI. The core idea of DGI is to learn how to distinguish between the nodes of two graphs. The DGI algorithm operates on the following bases:

- a true graph $\mathcal{G}$ with true nodes and edges, and
- a corrupted graph $\mathcal{H}$ in which the nodes and edges have been changed using a corruption function that can randomly shuffle node features while maintaining the same edges like those of the true graph $\mathcal{G}$ [6].

The training procedure of DGI training consists of four components:

- a corruption procedure $\mathcal{C}$ which changes a real graph $\mathcal{G}$ into a manipulated graph $\mathcal{H} = \mathcal{C}(G)$ by randomly shuffling its node features using a Bernoulli distribution,
- an encoder $\mathcal{E}$ which computes the node embeddings of a manipulated graph and a real graph using various graph representation methods (e.g., GCNs, GAT or GraphSAGE),
- a single embedding vector $\overrightarrow{\mathcal{S}}$ representing the whole graph computed by a read-out-function $\mathcal{R}$ that takes the average or the sum of all the node embeddings in a real graph, and
- a discriminator $\mathcal{D}$, which is a logistic Sigmoid non-linear function that compares a node embedding vector $\vec{h}_i$ with $\overrightarrow{\mathcal{S}}$ and provides a score between 0 and 1.

$$\mathcal{L} = \frac{1}{N+M}\left(\sum_{i=1}^{N}\mathbb{E}_{(\mathbf{X},\mathbf{A})}\left[\log\mathcal{D}\left(\vec{h}_i, \vec{s}\right)\right] + \sum_{j=1}^{M}\mathbb{E}_{(\tilde{\mathbf{X}},\tilde{\mathbf{A}})}\left[\log\left(1-\mathcal{D}\left(\vec{h}_j, \vec{s}\right)\right)\right]\right) \tag{2}$$

We need to maximise the score if $\vec{h}_i$ is the node embedding of a true node and minimise the score if $\vec{h}_i$ is the node embedding of a manipulated node to maximise the mutual information between patch representations and global graph summary. A binary cross-entropy loss objective function can be applied to train the model. After the training process, the trained encoder can generate new graph embedding for downstream purposes (e.g., node clustering).

4 Methodology and Approach

This study uses the Elliptic dataset, the world's largest Bitcoin graph transaction network with handcrafted features. In a transaction graph, the nodes represent transactions, and the edges represent transaction flows. This project uses DGI with GraphSAGE to cluster illicit and licit nodes in a realistic Blockchain transaction graph dataset.

4.1 Exploratory Data Analysis

The Elliptic dataset consists of 203,769 node transactions and 234,355 directed transaction payment flows. It also consists of graphs with 49 different timestamps uniformly spaced with a two-week interval.

21% of the node entities are labelled as licit in the Elliptic dataset, and 2% are illicit. The remaining node entities are unlabelled. These node entities consist of 166 features, among which the first 94 features contain local information of the transactions, including timestamps, transaction fees and the number of inputs or outputs. The remaining 72 features are aggregated features. Those features can be obtained by aggregating transaction information from one-hop backward/forward graph nodes, such as the standard deviation, minimum, maximum and correlation coefficients of the neighbour transactions for the same information data. More importantly, all features are obtained by using only publicly available information. Figure 4 shows the distribution of the number of node entities of different labels across different timestamps.

4.2 Graph Visualisation

To better understand transaction graphs, we visualise the graphs using Neo4j Bloom, which allows us to explore interesting patterns within the graphs [13]. Unlike traditional graph visualisation tools like NetworkX and Gephi, Neo4j Bloom provides a powerful NLP engine for querying graph data and visualising the results immediately. To perform graph visualisation, we first create a Neo4j graph database and then import the transaction data into the database. The command used to import the graph transaction data into the Neo4j database is given below, where the path to the *node* data is given by the nodes option, and the path of the *relationship* (edge) data is given by the relationships option:

```
$ bin/neo4j-admin import --database=AML --nodes=import/node.csv
↪ --relationships=import/edge.csv
```

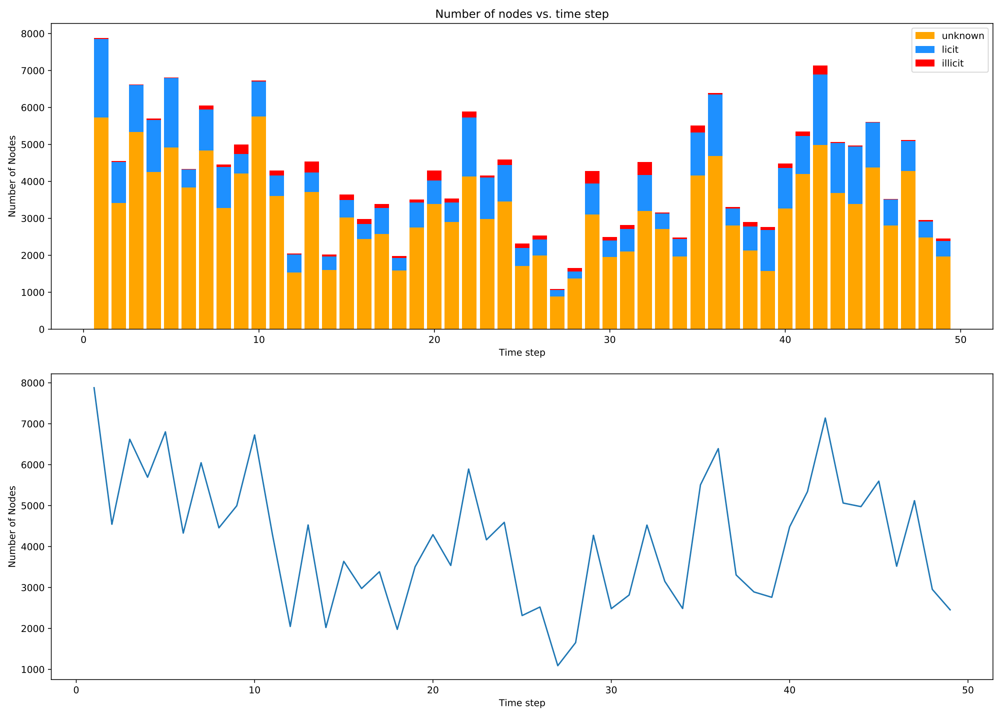
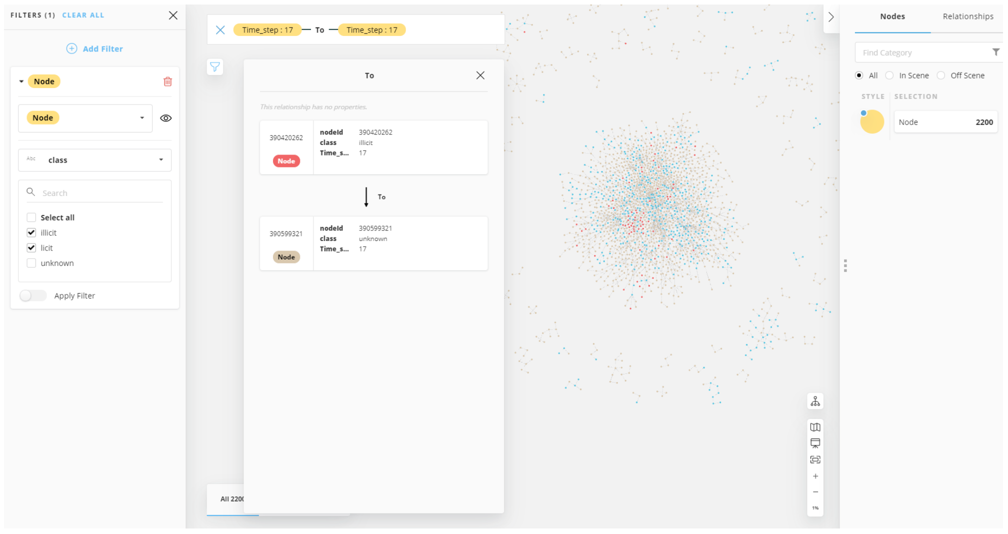


Figure 4: Number of nodes versus time steps

After importing the data into the graph database, Neo4j Bloom can be used for graph visualisation, fraud discovery, and analysis. The details of its use are shown in the demo section.



[ ] Select all
[✓] illicit
[✓] licit
[ ] unknown
Apply Filter

This relationship has no properties.

390420262	nodeId	390420262
	class	illicit
	Time_s...	17
390599321	nodeId	390599321
	class	unknown
	Time_s...	17

[✓ All] In Scene Off Scene

Node 2200

Figure 5: Sample Graph Visualization

4.3 Experimental Setup

4.3.1 Network Graph Construction

We constructed 49 different Bitcoin transaction graphs, one graph for each timestamp. The nodes represent transactions, and the edges represent flows of Bitcoin transactions. Figure 5 visualises 17 of the graphs constructed from the Elliptic dataset.

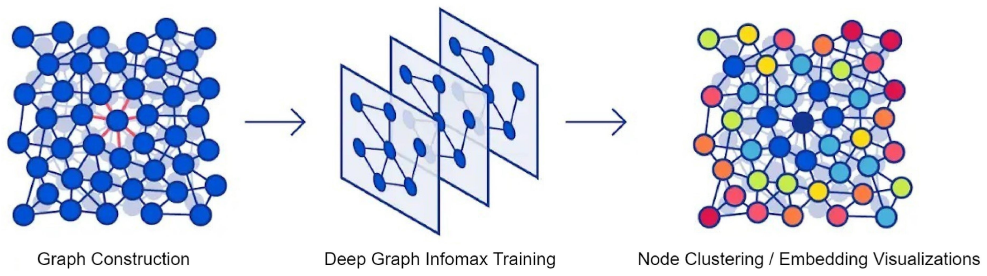

Graph Construction
Deep Graph Infomax Training
Node Clustering / Embedding Visualizations

Figure 6: Overall Experimental Process

4.3.2 Training of Deep Graph Infomax

In terms of the encoder $\mathcal{E}$, we use a GraphSAGE encoder with one-hop layer ($K = 1$) as DGI benefits from wider models, rather than deep ones [6]. We also use a mean aggregator to aggregate the information from one-hop neighbours. The mean aggregation function AGG is given by Equation 3 .

$$\mathbf{h}^{k}_{\mathcal{N}(v)} = \sum_{u \in N(v)} \frac{h^{k-1}_{u}}{|N(v)|} \tag{3}$$

Here $|N(v)|$ represents the number of sampled neighbour nodes at layer k, and $\mathbf{h}^{k-1}_{u}$ represents the neighbour's previous layer embeddings at layer $k - 1$. As to neighbour sampling, we specify the size of one-hop neighbour samples for GraphSAGE to be 30 and a hidden layer size to be 256 ReLU units for the GraphSAGE layers.

For the corruption procedure $\mathcal{C}$, following the approach of Veličković et al. [6], we randomly shuffle the node features of the nodes in a real graph $\mathcal{G}$ using a Bernoulli distribution to generate manipulated graphs. For the read-out function $\mathcal{R}$, we use the mean operation on node embeddings in a real graph to compute embeddings $\overrightarrow{\mathcal{S}}$ for the whole graph. For the discriminator $\mathcal{D}$, we introduce a logistic Sigmoid non-linear function which compares a node embedding vector $\vec{h}_i$ with the embeddings of real graphs $\overrightarrow{\mathcal{S}}$ to calculate the score of $(\vec{h}_i, \overrightarrow{\mathcal{S}})$ being positive or negative. After this, we apply a binary cross-entropy loss objective function to perform gradient descent as shown in Equation 2. Our experiments use all the 49 Bitcoin transaction graphs to train the DGI with the GraphSAGE encoder in an unsupervised manner. For each graph, we train two epochs using an *Adam* optimiser to perform gradient descent.

4.3.3 Node Clustering / Embedding Visualizations

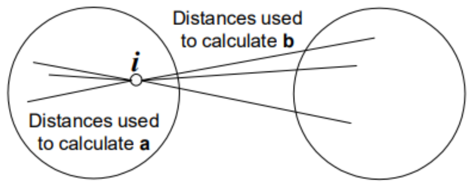

Distances used to calculate **b**
i
Distances used to calculate **a**

Figure 7: Silhouette Score [13]

After training the DGI, we extract the trained GraphSAGE encoder to generate node embeddings for node clustering and visualise the result of dimensionality reduction. Since the dataset consists of multiple transaction graphs, we select only the last two Bitcoin transaction graphs with timestamps 48 and 49. Their node embeddings are generated using the GraphSAGE encoder. Then, we apply the t-distributed stochastic neighbour embedding (t-SNE) dimensionality reduction algorithm to reduce the dimensionality of node embeddings to two so we can plot their embeddings on a two-dimensional plane. We also conduct dimensionality reduction on the raw features of the same graphs without applying any graph machine learning algorithm. Finally, we adopt the Silhouette score [13] to assess the clustering performance in terms of cohesion and separation.

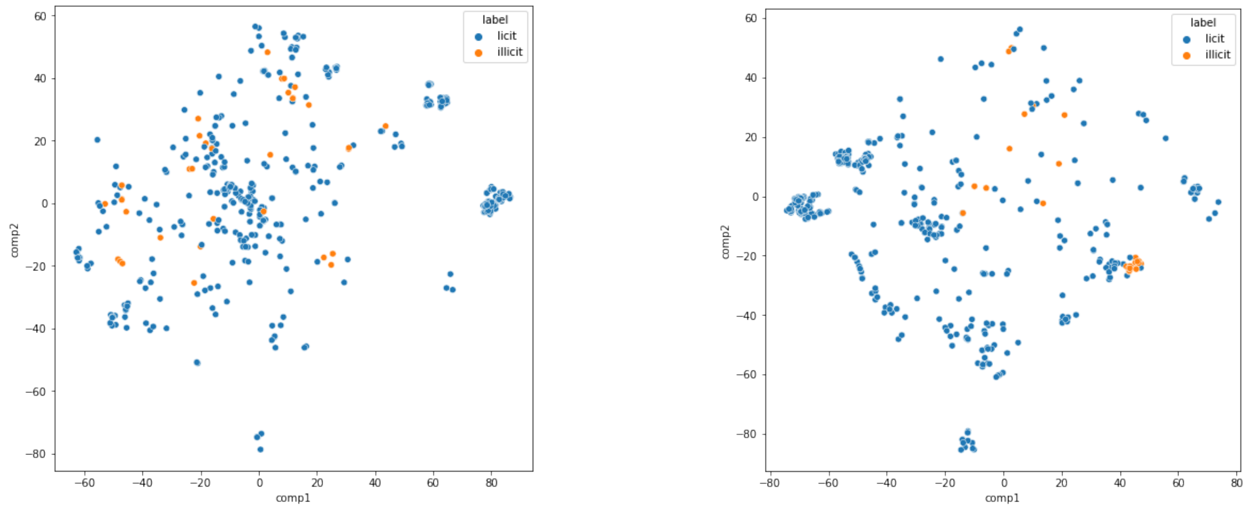


Figure 8: The result of performing dimensionality reduction on a) raw features of the transaction graph with timestamp 48, b) node embeddings of the same graph generated by DGI.

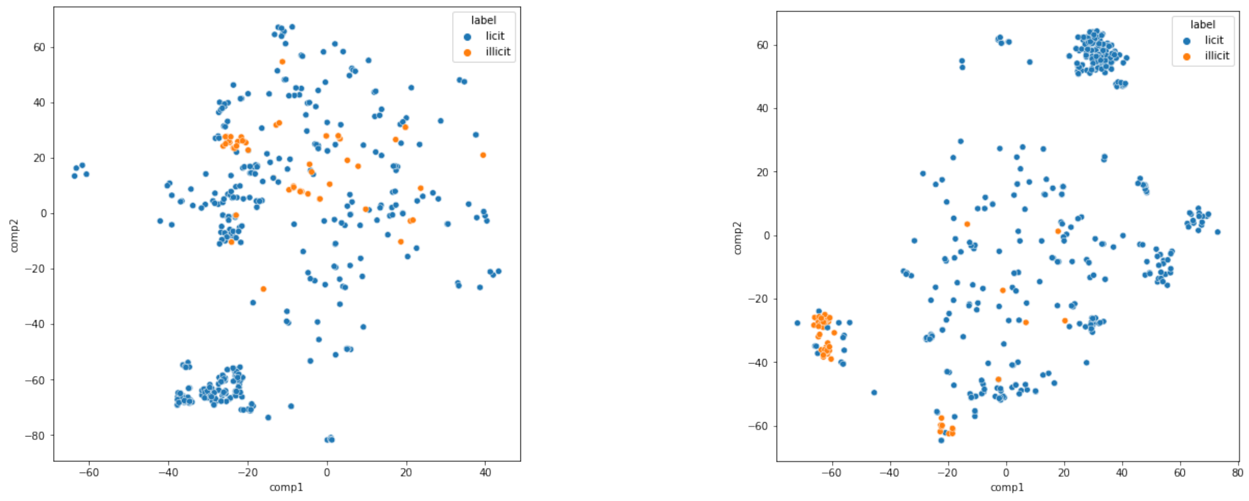


Figure 9: The result of performing dimensionality reduction on a) raw features of the transaction graph with timestamp 49, b) node embeddings of the same graph generated by DGI.

5 Experimental Results

To better understand the performance of DGI with a GraphSAGE encoder, we should investigate the embeddings generated by it. This study compares the node embeddings generated by the trained encoder with the raw features by visualising both. After converting the transaction data with timestamps 48 and 49 to graphs, we feed them to the trained GraphSAGE encoder to generate node embeddings. Then, we perform t-SNE on the node embeddings to reduce their dimensions to two and then visualise them. Similarly, we perform the t-SNE algorithm on the raw node features. To visualise node embeddings, we remove points of the unknown class. As shown by (a) and (b) of Figures 8 and 9, the embeddings are better separated than the raw data points. In terms of Silhouette scores, clustering DGI node embeddings achieves a score of 0.156 and 0.284 for graphs 48 and 49, whereas clustering raw node features achieves a score of -0.066 and 0.019. The clustering result suffices to show that our proposed algorithm can improve the performance of clustering licit and illicit transaction nodes.

6 Conclusion and Future Work

In this study, we developed a novel approach using DGI combined with a GraphSAGE encoder to detect illicit entities from the Elliptic dataset. Our results showed promising performance achieved by the proposed algorithm in clustering nodes in the latent space. In the future, it is important to perform a comprehensive analysis on different cryptocurrency datasets to fully explore the performance of DGI. This research opens the opportunity to explore more unsupervised graph machine learning algorithms for financial forensics.

References

[1] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Tech. rep. Manubot, 2019.

[2] Nir Kshetri and Jeffrey Voas. “Do crypto-currencies fuel ransomware?” In: *IT professional* 19.5 (2017), pp. 11–15.

[3] Mark Weber et al. “Anti-Money Laundering in Bitcoin: Experimenting with Graph Convolutional Networks for Financial Forensics”. In: *ACM SIGKDD International Workshop on Knowledge discovery and data mining*. 2019.

[4] Min Shi et al. “Multi-Class Imbalanced Graph Convolutional Network Learning”. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*. 2020.

[5] William L. Hamilton, Rex Ying, and Jure Leskovec. “Inductive representation learning on large graphs”. In: *Advances in Neural Information Processing Systems*. 2017. arXiv: 1706.02216.

[6] Petar Veličković et al. “Deep Graph Infomax”. In: *International Conference on Learning Representations (ICLR)* (2019).

[7] Yining Hu et al. “Characterizing and detecting money laundering activities on the bitcoin network”. In: *arXiv preprint arXiv:1912.12060* (2019).

[8] *Wallet Explorer*. URL: https://www.walletexplorer.com. (accessed: 01.09.2016).

[9] Thai Pham and Steven Lee. “Anomaly detection in bitcoin network using unsupervised learning methods”. In: *arXiv preprint arXiv:1611.03941* (2016).

[10] *Blockchain Explorer*. URL: https://www.blockchain.com/ko/explorer. (accessed: 01.09.2016).

[11] Ismail Alarab, Simant Prakoonwit, and Mohamed Ikbal Nacer. “Comparative Analysis Using Supervised Learning Methods for Anti-Money Laundering in Bitcoin”. In: *Proceedings of the 2020 5th International Conference on Machine Learning Technologies*. ICMLT 2020. Beijing, China: Association for Computing Machinery, 2020, pp. 11–17. ISBN: 9781450377645. DOI: 10.1145/3409073.3409078. URL: https://doi.org/10.1145/3409073.3409078.

[12] Jie Zhou et al. “Graph neural networks: A review of methods and applications”. In: *AI Open* 1 (2020), pp. 57–81. ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2021.01.001. URL: https://www.sciencedirect.com/science/article/pii/S2666651021000012.

[13] Peter J. Rousseeuw. “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53–65. ISSN: 0377-0427. DOI: https://doi.org/10.1016/0377-0427(87)90125-7. URL: https://www.sciencedirect.com/science/article/pii/0377042787901257.